Cloud Integration
Technical Whitepaper

IBM API Connect Deployment Whitepaper

IBM

Contents

1 Abstract 3

2 Introduction 3

2.1 IBM API Connect Overview 3
2.1.1 Management server 3
2.1.2 API Gateway 4
2.1.3 Developer Portal 4
2.2 Terminology 4
2.3 Cross-component data flows 5
2.4 Relationship between logical concepts and deployment components 7
2.5 Environment separation 8
2.5.1 Shared Management service 8
2.5.2 Gateway service 9
2.5.3 Developer Portal 11
2.5.4 Recommended environment separation 11
2.5.5 Recommended practice for API development and promotion 12

3 Assessing your quality of service requirements 14

3.1 Understanding key user scenarios for API Connect 14
3.2 Failure scenarios 14
3.3 Questions for identifying availability requirements 15

4 Typical deployment patterns 16

4.1 Minimal development install 16
4.2 Single region with high availability 17
4.3 External and internal API exposure 18
4.4 Dual region with high availability 20
4.5 Bluemix Dedicated deployment 21
4.5.1 Single environment Bluemix Dedicated 22
4.5.2 Two environment Bluemix Dedicated 23
4.6 Hybrid gateway topology 24
4.7 Global deployment for geographical affinity 25
4.7.1 Reduced topology global deployment 26

5 Key points for multi-region deployments 27

5.1 Deployment components 27
5.2 API Gateway 28
5.2.1 Resilience to failure of other tiers 30
5.3 User interface traffic 31
5.4 Management tier 32
5.4.1 Configuration database 33
5.4.1.1 Mitigating Split Brain Scenarios with “Main Site” 34
5.4.2 Analytics repository 35
5.5 Developer Portal 38
5.5.1 Failover and cluster sizing 39
5.6 Business services 41
5.7 Loopback runtime 41
5.8 Supporting components 42
5.9 Example deployment topology 42

6 Operational topics 45

6.1 Disaster recovery using backup and restore 45
6.1.1 Management and Developer Portal 45
6.1.2 Analytics 46
6.2 Custom branding for APIs and Developer Portal endpoints 47
6.2.1 API call branding 48
6.2.2 Developer Portal branding 48

7 Summary 49

7.1 About the author 49

1 Abstract

This whitepaper describes the key concepts and topics you will need to understand to deploy IBM API Connect v5 for use in a production scenario. We will discuss the various topologies that are typical for API Connect and the things that need to be considered to ensure that your deployment will successfully meet the relevant requirements for availability, scalability, resilience and other aspects related to providing a production quality of service for the solution.

The paper focuses largely on the on-premises API Connect form factor which is deployed and administered by your team, however there is also discussion of some aspects of the Bluemix offerings where IBM is responsible for the deployment and administration and you consume the offering "as a service".

2 Introduction

2.1 IBM API Connect Overview

IBM API Connect is an end-to-end solution that quickly and easily allows you to create, run, secure and manage access to APIs. It provides a range of powerful capabilities through which you can discover or define your API, implement that API to connect with your existing backend services or create new microservices to expose data assets. API Connect also enables you to expose your APIs to your target application developers whether they are inside or outside your organization through a self-service Developer Portal that means new users of your APIs can be up and running within minutes.

In this whitepaper, we will discuss the deployment and administration aspects of API Connect and the way in which you deploy the offering as part of your production infrastructure. The following diagram shows the deployment components that make up the API Connect solution.

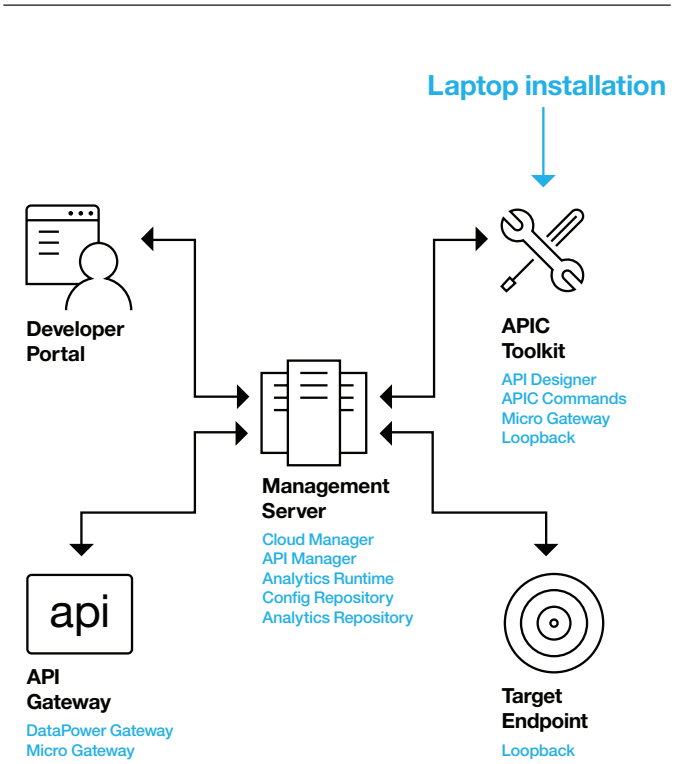


Figure 1: **Deployment components**

2.1.1 Management server

The Management server is the central coordinator of the whole solution;

- It hosts the Cloud Manager user interface through which the system administrator deploys and configures the solution
- It hosts the API Manager user interface that the API provider users will use to define and configure the runtime Catalog into which they will deploy APIs and manage their lifecycle
- It contains a persistent database that is used to store the configuration data about the system including details about;
  - Users and their permissions
  - Catalogs, Products and the APIs they contain
  - Developer Organizations, the Applications they own and the Subscriptions that register those applications as having access to Products
- It contains the Analytics component which persists data about the set of API calls that have been invoked so that they can be queried by the user to understand how successfully the APIs are being used

2.1.2 API Gateway

The API Gateway is the core runtime component of the system – it is responsible for responding to incoming API calls from applications;

- Validating that the application that made the API call is permitted to access the API (i.e. has an active subscription to a product that contains the API operation)
- Enforcing the security constraints defined by the API such as a requirement to authenticate using a protocol like Basic Authentication or OAuth 2.0
- Enforcing rate limits so that the calling application cannot invoke the API more frequently than the API provider has specified
- Invoking the outbound request to the backend service or services that are defined in the API implementation, which may involve protocol transformation such as a REST API calling out to a backend SOAP service
- Aggregating responses from potentially multiple backend service calls and returning the relevant content of those requests to the original caller

The API Gateway function might be provided by the DataPower gateway - which is common in public or partner scenarios where the gateway is hosted in the DMZ, or by the MicroGateway – which is a lightweight Node.js based gateway which is suited well to internally facing scenarios, particularly where the administration of the gateway component itself will be handled by the same project- or line-of-business team that manages the APIs that it serves.

2.1.3 Developer Portal

The Developer Portal is the component through which application developers that wish to consume the APIs will access the system, including;

- Discovering the set of APIs that are made available by an API provider
- Self-service registration of their own developer organization account so that the application developer can authenticate themselves to the system
- Self-service creation of an application identity through which they will be allocated the Client ID and Client Secret credentials that are required to identify their application when making API calls
- Ability to subscribe the registered application to an API Product to grant it permission to invoke APIs (or request that they be granted that access, if the provider of the API has decided the provider must approve subscription requests)

The provider of the Developer Portal can customize the look and feel of the Portal so that it matches the corporate branding requirements of the enterprise, as in many cases the Developer Portal will be the public face of your API program.

2.2 Terminology

In this whitepaper, the following definitions of terms are used;

Term	Definition
Region	A physical location where infrastructure may be hosted in a way that it will not be affected by issues in other locations. Commonly also described as a site or data centre, with its own independent power and networking connectivity etc. A region may or may not have further sub-isolation characteristics such as semi-independent pods or availability zones.
High availability	The ability of the solution to continue successful operation in the event of failure of a subset of the system components, preferably without manual intervention.
Disaster recovery	The process of recovering the successful operation of the solution in the event of a total loss of the current infrastructure - for example recovering from backup into a new region.
Hot standby	A deployment configuration in which a set of servers are actively running ready to instantaneously take over in the event of a failure, but not actively serving traffic until that failover.
Cold standby	A deployment configuration where the failover servers are in a stopped state until the point a failover is required, resulting in a longer time window required to restore the normal operation of the solution.
Management service	In API Connect, the name given to the cluster of Management servers that makes up the control plane of the solution.
Gateway service	In API Connect, the name given to the cluster of Gateway servers that handle incoming API call traffic for deployed APIs.

Table 1: **Terminology**

Note that there are different interpretations about the meaning disaster recovery compared to high availability depending on the scope of "disaster" that you are referring to. Some customers deploy a solution across two data centres in a hot standby configuration and use the phrase "disaster recovery" to refer to the process of failing over to the second site in the event of the failure of the first site, however in this paper we describe that as one of the cases of high availability, and use disaster recovery to refer to the loss of **all** the deployed infrastructure.

2.3 Cross-Component Data Flows

The following diagram illustrates key data flows between the various components of the system and their respective users. Understanding the connections between the various components of the system helps to position many of the requirements for deploying IBM API Connect successfully for high scale usage.

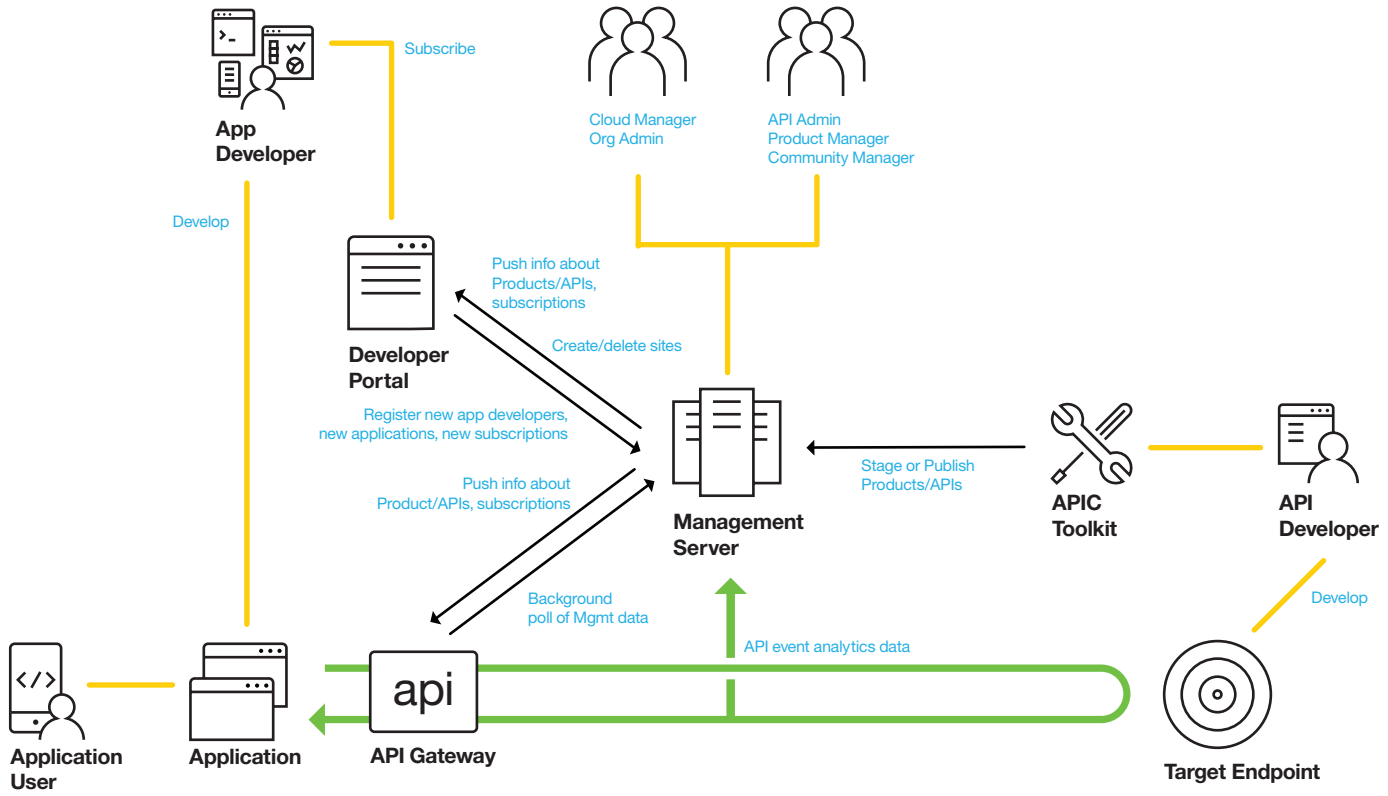


Figure 2: **Cross-component data flows**

User interactions

The diagram above highlights several user roles and the interactions that they have with the system;

- Cloud Manager role — the system administrators who deploy and configure the infrastructure that makes up the solution. These users work mainly in the Cloud Manager user interface or with the appliance command-line.
- API Developer — designs and implements the APIs that will be exposed through the product. Works mainly with the API Connect Toolkit installed on their laptop
- Product Manager — responsible for publishing API Products to a Catalog when they are ready to be consumed. Works mainly in the API Manager user interface on the Management server, or with the API Connect Toolkit command-line interface
- Community Manager — manages the set of Developer Organizations, Applications and Subscriptions that consume the published Products. Works mainly in the Community section of the Catalog in the API Manager user interface
- Application Developer — implements the Application that consumes the APIs. Discovers and subscribes to APIs through the Developer Portal and then writes the application that invokes the APIs through the API Gateway

Each of these users contributes to different stages in the API's lifecycle and has discrete actions they need to carry out to achieve those goals. The availability expectations they have of the system will be different, and dependent on the actions that they perform.

System interactions

The components of the system exchange data at various points to perform their necessary functions, largely coordinated by the Management server. For example;

- When a Product is published to a Catalog by the API Developer using the API Connect Toolkit the Management server will forward a copy of the necessary parts of that API to both the Developer Portal and the API Gateway so that they can display the details about that API or serve incoming API calls for it, respectively
- When an Application Developer logs in to the Developer Portal and subscribes one of their Applications to a new Product the details about that new subscription are sent to the Management server where they are persisted, and subsequently forwarded to the API Gateway so that the gateway will accept incoming API calls under the new subscription

2.4 Relationship between Logical Concepts and Deployment Components

The following describes the logical concepts that make up the API Connect solution;

Concept	Description
Cloud	A deployment of API Connect consisting of a single Management service (cluster) and associated Gateway and Developer Portal services.
Provider organization	Owns a set of APIs that will be deployed and managed through API Connect. Multi-tenancy support at this level which means there may be many different provider organizations – for example one per project, line of business or team. Each provider organization is logically isolated from the others with a different set of users that can configure or manage the APIs that it contains.
Catalog	Each provider organization will have one or more Catalogs that are the place to which Products (containing APIs) will be staged or published. There may be multiple catalogs in a provider organization to reflect the dev/test/pre-prod/prod stages of the API process, or potentially to segregate different sets of APIs for different consumers, including externally facing consumers or internally facing consumers.
Developer Portal (site)	Each Catalog can have its own Developer Portal site configured, which is a website that application developers will use to discover Products and subscribe to them. The site can be customized to have whatever look and feel the owner wishes, and typically adopts the corporate branding of the enterprise that exposes it to fit in with the other public facing websites.
Gateway service	Each Catalog is associated with a Gateway Service, which is the cluster of API Gateway servers that will be instructed to serve the API requests for that Catalog.
Developer organization	A developer organization exists within a Catalog as the representation of a business entity that wishes to consume APIs that are exposed by the Catalog. For example, a 3rd party application development company would register themselves as a developer organization (via the Developer Portal). Each developer in their company can then be registered as a user inside that developer organization.
Application	One or more Applications are registered by a Developer Organization to subscribe to APIs. The application will be allocated the Client ID and Client Secret credentials that it will supply when invoking API calls.

Table 2: **Selected logical concepts**

Each deployment component shown in **Figure 1** is deployed as a cluster of one or more server instances depending on the availability and resilience requirements of the deployment, however the solution is designed with the ability to support many different projects or teams sharing the same deployed service instances;

- The Management service can support hundreds of provider organizations, although in a typical on-premises deployment there will generally be less than 20 - representing individual project teams or lines of business
- The Management service can also support many hundreds of catalogs, which at this level are only a logical subdivision of the provider organization
- The Developer Portal service can support hundreds of catalogs — each catalog will have its own site created on the Portal cluster. For example, a cluster of three Developer Portal servers might host 30 portal sites across 18 provider organizations
- A single Gateway service can also support hundreds of catalogs, however it is also possible to define multiple Gateway services if you wish to segregate traffic for different catalogs — for example a set of public facing catalogs exposed through the DMZ in one Gateway service, and internally facing catalogs using a second Gateway service
- Developer Organizations and Applications are logical subdivisions within a catalog – in public facing API scenarios it is common for there to be many hundreds or thousands of these items within a catalog

2.5 Environment separation

The design, implementation, validation and deployment of a new API to production is a process much like that of a conventional business application — there are various checkpoints that the API will go through before it is released for use by real application developers, and your API Connect deployments need to support that lifecycle.

It is common for customers to have a sequence of "environments" in their deployment process, with the expectation that the API infrastructure can support each of those steps in an appropriate fashion, for example;

- Development
- Functional test
- Performance or Load test
- Pre-production / staging
- Production

API Connect provides a spectrum of options for isolating or sharing various components of the system which can help to strike a balance between fully isolated copies of the entire deployment (which is best for isolation but has a higher impact on administration overhead and licensing costs) down to implementing all the environments with a single deployment (low administration overhead but poor for isolation).

2.5.1 Shared Management Service

The Management service is synonymous with the scope of the "cloud", so if you want full isolation between two environments then they must have different Management services — which means a different set of Management servers for each environment.

Sharing a Management service means there will be a single set of administrator users who have access to the Cloud Manager interface for carrying out administration activities — therefore there is commonly a separate Management service for **at least** the Production environment (often in an isolated network) since not everyone in the development team will have a business need for access to the Production environment.

As you saw in the previous sections the Management service also handles the storage of API event analytics data for all the Gateway servers defined underneath it, so there is a level of "runtime" impact for the Management service. As a result, any environments that expect high API call throughput (such as Performance or Load test) should lean towards having their own separate Management service in order reduce the potential for the high API call traffic rate to affect other environments.

The Management service is also the key component in the process of upgrading API Connect from one release to another or for applying Fix Pack updates within a version – the cluster is always at a single version of API Connect and so if you upgrade the Management cluster then you are recommended to update all the related tiers to the matching version at the same time. This can introduce friction if you deploy multiple Gateway services to serve different types of traffic as there is then a requirement to upgrade all the Gateway services at the same time.

2.5.2 Gateway Service

The Gateway service is a cluster of servers that are configured so that they serve the same set of API calls. For full isolation, a different set of servers should be defined inside each Gateway service, but the DataPower gateway does provide the ability to add the same servers to multiple Gateway services.

The effect in that case is to deploy a different DataPower "domain" for each Gateway service that the server has been added to, which provides a level of administrative isolation between the APIs deployed to different Gateway services even though the same DataPower servers are being used for both.

Each domain must be bound to a different combination of network interface and IP address/port on the DataPower server so this technique of using the same DataPower servers in multiple Gateway services is often useful in the following circumstances;

- Allow different catalogs to be bound to different network interfaces - for example a public facing NIC, and an internally facing NIC in the case where the DataPower server spans multiple network zones
- Separate the data traffic for different environments, such as Development traffic being routed over a different NIC or port number. This might be used in conjunction with the configuration of networking rules to ensure that "development APIs" can't contact "staging endpoints" or similar

Note that the different Gateway services do not need to be defined within the same Management service – you might have isolated Management services (for other reasons) and then choose to share the same DataPower servers, to increase the utilisation of those servers – but at the risk of increasing the potential for one cluster to affect the behaviour of the other.

Importantly although different DataPower domains are used for each Gateway service it is not currently possible to pin resources to a given DataPower domain, so you can't (for example) divide the CPUs assigned to the DataPower server across the two domains. Thus, there is a "noisy neighbour" scenario where exceptionally high traffic in one Gateway service could impact performance of the other, and so sharing of DataPower servers is not recommended for performance-sensitive scenarios.

The following diagrams illustrate the two cases—the first diagram shows different DataPower servers as members of each Gateway service so that there is no runtime impact of the operation of Gateway service A on Gateway service B. The second diagram has the advantage of using half the number of DataPower servers, but at the risk of degradation in performance of one Gateway service due to high throughput in the other.

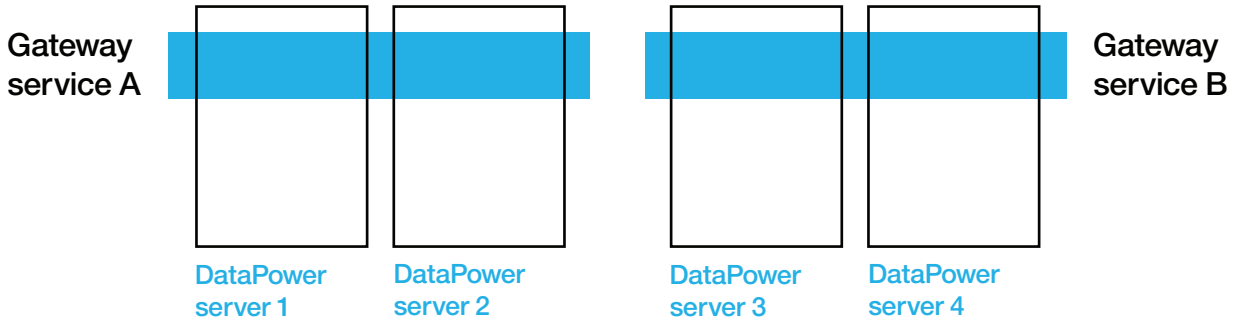


Figure 3: **Two gateway services that do not share DataPower servers**

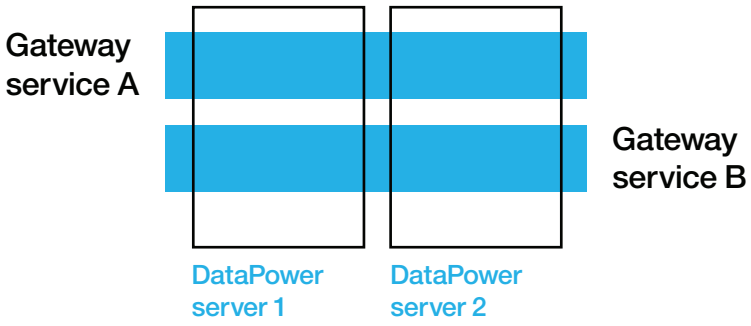


Figure 4: **Two gateway services that share DataPower servers**

For additional information on this topic see the Knowledge Center topic on "Adding multiple Gateway services that share a single DataPower appliance".
ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.cmc.doc/create_multiple_gateway_clusters.html

2.5.3 Developer Portal

Currently it is necessary to deploy separate Developer Portal servers for each Management service - it is not currently possible to share Developer Portal servers across multiple Clouds. Also, there is only one Developer Portal cluster for a Management service (cloud), so you cannot use different sets of Developer Portal servers for externally facing catalogs than internal catalogs.

As noted in section 2.4 however a single cluster of Developer Portal servers is capable of hosting Portals for many different Catalogs at the same time.

2.5.4 Recommended Environment Separation

The specific set of environments used by each customer is often subtly different and so there is no single answer to the question of how to lay out the infrastructure for your environments but the following guidelines are relatively common in their use by customers. Each of the top-level bullets here represents a separate Management service, typically with isolated resources within the environment for the other tiers.

- Dev/test
  - A single Management service for both these purposes
  - Use different catalogs within the provider org to separate development from test
  - Some customers have multiple development or test catalogs if they wish to sub-divide further
  - Typically have a segregated set of Gateway instances for dev/test use (separate from other environments), but generally a single Gateway service unless there is a need to isolate traffic between development and test at a network level

- Performance or load test
  - True performance test environments should be fully isolated from other infrastructure to give reliable, repeatable results (ie no sharing of Management service or Gateway services)
  - Some customers omit having a separate deployment here and choose to use the dev/test cloud, with a policy that normal development and testing is suspended while performance testing takes place
  - As with all performance/load testing it is important to estimate the peak requirements of the deployment, which may be driven by seasonal or regular events such as vacation seasons, end of month, end of enrolment deadlines etc, and then test against an appropriate margin in excess of the expected traffic volumes so the size of the infrastructure for this environment will mimic Production

- Pre-production / staging
  - Ideally this would be a direct copy of the Production deployment (see below) to flush out as many issues as possible before they reach production
  - In some cases pre-production might be implemented as a separate logical catalog in the production deployment as long as the potential interference between the two is understood and accepted

- Production
  - Separate Management service
  - Fully isolated infrastructure, often in a protected network zone (compared to Dev/test etc)
  - Restricted user access – deployments may be carried out in an automated fashion using continuous delivery pipelines like UrbanCode Deploy or Jenkins

2.5.5 Recommended Practice for API Development and Promotion

A common question that follows from the separation of environments like Dev, Test, Pre-Production, Production is how it is recommended to manage the progression of APIs through those various environments. As with other developed components like applications there is a desire for the process to be simple, reliable, reproducible and auditable – particularly towards the end of the environment progression where the Operations or production Administration team wants to be able to roll out the updates to Production with a single automated click rather than having to work through a series of manual steps.

IBM API Connect v5 introduces important capabilities that support this automated promotion of APIs through the lifecycle in the form of the Developer Toolkit Command-Line Interface (CLI) which provides a series of scriptable commands for automating the deployment and management of Products in a Catalog.

When developing an API or Product the API developer is recommended to use the API Designer – a graphical interface within the Developer Toolkit that provides the ability to create, edit and test APIs locally on the developer's laptop without having to interact with the API Connect runtime components (such as the Management, Gateway or Developer Portal servers).

This "offline" development experience also has the advantage that the files representing the Product, the APIs and all their related artifacts are stored locally on the filesystem and so can then also be integrated with an external source code control system of your choice, which then becomes the basis for a continuous deployment pipeline in which the API can be automatically deployed to the relevant runtime catalog.

The following diagram illustrates the typical flow for implementing a new API;

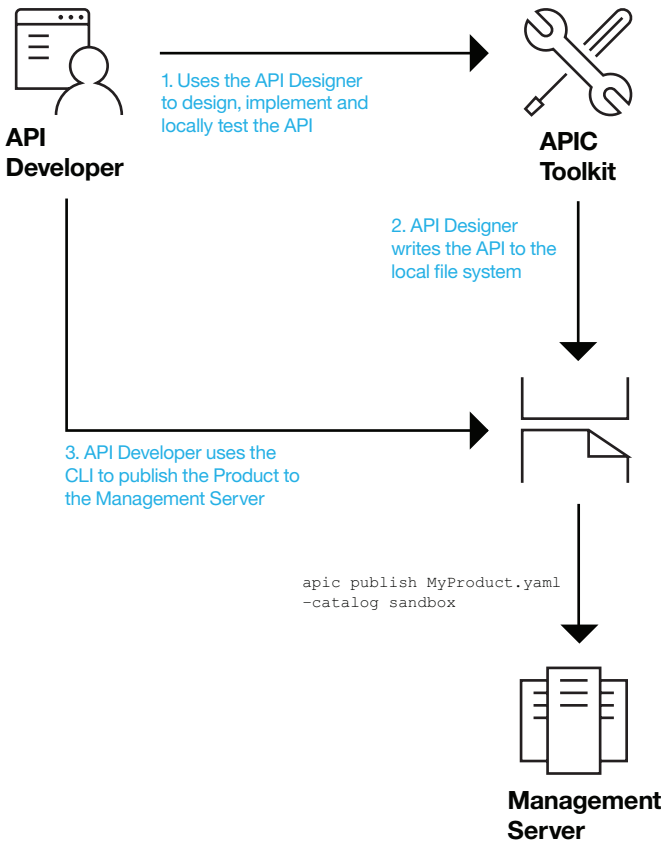


Figure 5: **Developing an API using the offline Developer Toolkit**

Note that the API Developer is publishing the Product directly to the Catalog in the Management server – they are not making use of the "Drafts" view that can be found in the API Manager. This "Drafts" experience is primarily intended as a backwards compatibility feature with the previous IBM API Management v4 release, in which all development of APIs was done "online" in the API Manager. In version 5 the recommendation is for API Developers to use the Developer Toolkit to develop their APIs locally on their laptop and not to make use of the Drafts view.

Once the API developer has finished their implementation and testing work they will be ready to submit the new version of the Product / API to the next stage in the delivery pipeline, which might be "Test" or similar. It is at this point that customers often start to introduce an automated delivery pipeline to manage the evolution of the API in a more formal manner.

The automated delivery pipeline typically includes the following steps as shown in **Figure 6** below;

1. API developer commits the files representing the changes they have made from their local filesystem to a source code control system
2. A plugin to the source code control system provides the ability to trigger a notification (or a job) when file changes are detected
3. A job executes which extracts the source code onto the job server
4. The job executes a script which uses the Developer Toolkit CLI to publish the Product to the target API Connect cloud
5. The job executes a series of functional verification tests
6. If the tests are successful the job might then execute the next job in the cycle, which might deploy and test the changes to the Pre-Production cloud

Note that API Connect does not mandate or depend on any specific choice of tooling, so although Git and Jenkins are shown in the example above you could use any source code management (SCM) tool or job execution infrastructure that is in place in your enterprise – the Developer Toolkit CLI provides the necessary functionality in a way that it can be embedded in any technology you choose.

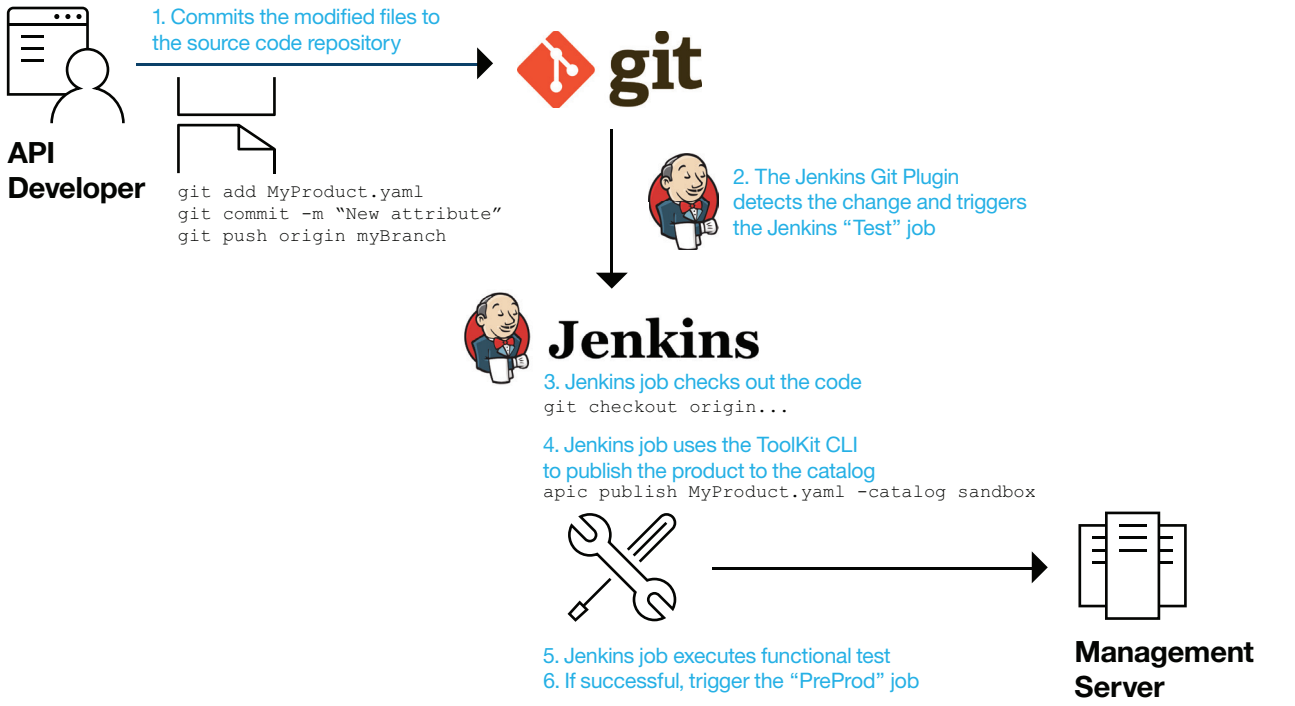


Figure 6: **Automated delivery pipeline example**

3 Assessing your Quality of Service Requirements

3.1 Understanding Key User Scenarios for API Connect

An easy trap to fall into when planning any production deployment is for you or your stakeholders to make a statement that the infrastructure must be 100% available. We would all like our services to be available 100% of the time but achieving that level of availability comes with a financial and personnel price tag that not many organizations are willing to pay when it comes to the crunch!

Instead we need to have an honest and open conversation with our stakeholders to understand what the practical availability requirements of the system are, and what the implications are if those requirements are not met. We also need to take into consideration that there are typically different availability requirements for different data flows and user scenarios within a deployment, which map into constraints on the infrastructure that supports those scenarios.

In the context of API Connect the following represent the key scenarios that you will want to understand from the perspective of their impact to the business if there is a failure;

1. **Deployed API calls** — the ability to invoke APIs that have already been deployed to the platform
2. **Developer Portal viewing** — ability for application developers to view the documentation for existing APIs
3. **Developer Portal updating** — ability for new application developers to register, or for app developers to subscribe to products
4. **Analytics collection** — ability to collect new analytics event data about incoming API calls
5. **API Manager** — ability to manage existing APIs or deploy new ones through the user interface or command-line toolkit
6. **Analytics query** — ability to retrieve/view historical data about API events

The set of items above is listed in a typical order of decreasing priority, in that the highest importance aspect of the product functionality to many customers is the availability and correct functioning of existing deployed API calls. This is the bedrock on which you are running your business and is the most important part of the system so it typically sets an upper bound on the availability requirements of the other scenarios.

For a public or partner facing API programme the next most important thing is likely the Developer Portal since that is also a public facing interface to your infrastructure. As shown above it is useful to consider two variations of scenario around the Developer Portal – the first "read only" view, and the second "update" scenario. This acknowledges that Application Developers will often just be browsing your API catalog looking for useful services before they get to the point of registering and subscribing to your APIs.

The API Manager interaction scenarios are often significantly lower down the priority order compared to things like deployed API calls because in a production environment new Products or APIs are typically rolled out relatively infrequently and there is a lower impact if a rollout must be delayed for a period due to an outage of part of the infrastructure. These scenarios become more important however if you are implementing true continuous delivery where you may wish to publish new APIs multiple times a day rather than only a couple of times a month.

The requirements on the availability of the Analytics aspects of the service are directly linked to what use you intend to make of the data. If you are monetizing your API programme then having a reliable record of how many API calls have been made by a given application has a direct link to your revenue, however if the analytics data is only for informational purposes then it may not have a significant impact if you miss blocks of data for a given period or are unable to query it immediately.

3.2 Failure Scenarios

There are a range of potential failure scenarios with any infrastructure deployment and it is important to establish which of them you need to be able to survive to meet your business requirements. For high profile projects the answer may well be "all of them" but in other cases it may be acceptable that the service goes offline for a while because the stakeholders have agreed that the cost to implement support for continuing service during that scenario outweighs the cost to the business given the expected frequency of the event.

The following lists a range of potential failure scenarios – in each case you should consider how frequent the event to be, what the impact would be to your service or business, and how much effort/money you would be willing to invest to prevent it from impacting your service;

1. Failure of a single node in an active data centre
2. Communication failure between two data centres
3. Loss of all function for a given component within a data centre
4. Loss of multiple instances across one or more data centres(?)
5. Loss of an entire data centre

It is also important to consider the impact of the other services that API Connect depends upon, and what failure scenarios they could trigger. One of the key dependencies that API Connect has are the backend services that are being managed by our APIs – there is limited value in deploying a multi-region highly available infrastructure for API Connect if the backend service is only present in a single region and has a worse availability profile than the API endpoint you are trying to expose;

- Where are your backend services located?
- What are the failure characteristics of those services?
- Are those backend services replicated in both data centres so that it continues to function if one data centre fails?

3.3 Questions for Identifying Availability Requirements

To reach an accurate understanding of your availability requirements it is often useful to survey your stakeholders with concrete questions that help them quantify the impact of availability on their business scenarios. Since the most common response to questions about availability will be "Yes – the system should always be available" we need to phrase those questions in a way that enables the stakeholder to give a more granular answer that quantifies the impact on a scale rather than just giving a binary yes/no response.

The following questions are useful starting points in the dialog;

1. What is the impact to your business if your API calls are down for 5 minutes?
   - What is the direct financial cost to your enterprise?
   - What is the reputational cost?
   - How does that impact change if the outage were 1 hour, 3 hours, 6 hours or 12 hours?
2. Who are your target application developers?
   - Are they internal / external to your organization?
   - Do they work "normal" business hours?
   - What time zone(s) are they located in, relative to your deployment?
3. What is the impact to your business if application developers are unable to view your API documentation for 1 hour?
   - What about 3 hours?
   - What if they could view the details of your APIs but couldn't subscribe to them?
4. How often do you expect each of the failure scenarios to occur?
   - What is the cost-benefit analysis of the business impact of each event versus its predicted frequency?
5. How important is the analytics capability to your usage?
   - Are you using analytics for chargeback/billing?
   - What is the business cost if you were unable to recover historical analytics data?
   - How much revenue would you lose per hour of API call outage?
6. Do you need all components of the system to be available in all locations?
   - If you have a three-region deployment does the Developer Portal (for example) need to be available in all three locations, or is it acceptable to run using a single region in the event of a total outage of one region?
   - Might it be acceptable that the Developer Portal only run in a single region, for example if the application developers are internal to your organization and have lower requirements on availability
7. How frequently do you expect to use each of the components of the system?
   - How often will you be deploying new APIs?
   - How often will you be registering a new Application Developer through the Developer Portal?
   - How often will an Application Developer register a new subscription?
8. For multi-region deployments, do you want to maintain the affinity of requests within a given region?
   - This may be more efficient from a latency perspective
   - What if a particular component within a region has a failure?

4 Typical Deployment Patterns

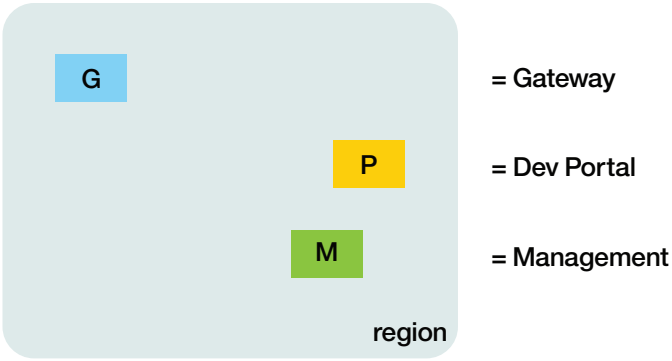
There are a series of common deployment topologies that we see being used across a wide range of customers. This section describes those scenarios and highlights the benefits and drawbacks to each of them against the goals that customers typically have in mind.

The patterns are listed in approximately increasing order of complexity starting with the simplest case and steadily building up more powerful topologies to meet additional deployment requirements. The contents of this section are not intended to be taken as an exhaustive list of supported scenarios – some customers mix-and-match certain aspects of the scenarios described below or use similar deployment topologies but with different requirements and motivations behind their decisions – the goal is to describe the common scenarios and reasoning that enable you to make an appropriate decision about your own deployment topologies.

4.1 Minimal Development Install

The first deployment most customers undertake is a small "development" installation. In the short term this gives your Administration team the opportunity to become familiar with the API Connect deployment components and how they sit within your corporate infrastructure, but the main intention of this style of deployment is to give your API developers a sandbox in which they can experiment freely with APIs, their implementation and usage without the risk of affecting users who require a high level of availability or stability such as production services.

Typically, a development installation will be contained within a single region (data centre) and does not have strict requirements on availability, so can be achieved using a single instance of each deployment component as shown in the following diagram;



Note: Loopback infrastructure omitted for illustration purposes

Figure 7: **Minimal development install**

This style of deployment is well suited for Proof of Concept or simple functional validation scenarios in that it provides all the capability and functionality of the API Connect offering, without the need for the high availability or resilience aspects. This means that it is not necessary to configure clusters of multiple servers for each of the tiers (a single server may suffice) and as a result it isn't necessary to set up load balancing in front of the clusters to route around failures.

For development or test installations it is common for the API developers to have an increased level of access to administer the deployment compared to one of the more formal environments like Production or Pre-Production. One example is where the API developer might be given command-line and/or browser based access to the DataPower instance (Gateway) for them to develop and test custom policy implementations, or carry out low level debugging of API call issues using the native DataPower capabilities. This more permissive approach to access can help to make the API development team more agile and productive in that they can investigate and resolve problems themselves without having to engage one of the system administration team, and so ensures that API projects can be delivered more efficiently in a faster time and with higher quality.

For some customers, the "development" environment still comes with expectations and requirements around availability and resilience and so the one-server-per-cluster approach described above may be insufficient. That decision is often determined based on the scope of impact an outage of the system would cause – if the development environment is only being used by a small number of developers in a single team then the one-server-per-cluster approach may be acceptable, but if the same development environment is shared by multiple distinct teams or many API developers then the collective loss of productivity from an outage may warrant either a single resilient clustered deployment (as described in the next section) or multiple independent deployment environments that can each be used by a single team.

4.2 Single Region with High Availability

Adding high availability for API Connect within a single region is a simple case of deploying multiple instances of each component and configuring appropriate load balancing in front of each cluster to route traffic around failures of any individual component as shown in the following diagram.

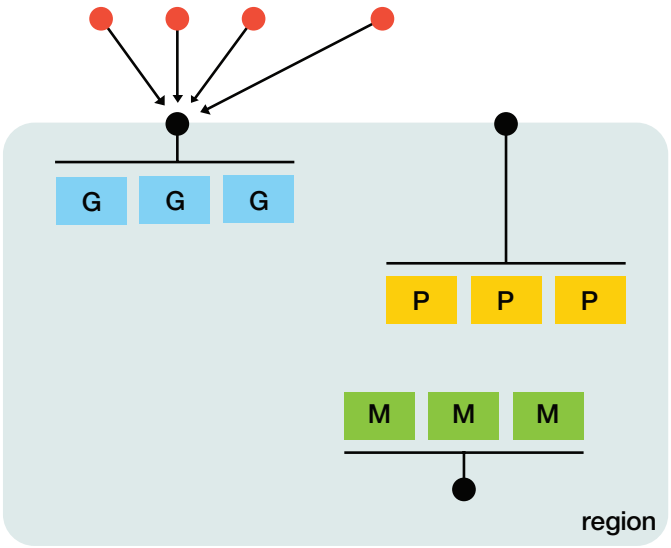


Figure 8: **Single region with high availability**

The action of adding additional servers to the cluster (for example adding servers to either the Gateway service or the Management service via the Cloud Manager) is all that is required to configure the necessary clustering and data replication within the service – all the necessary setup will be carried out automatically on your behalf without the need for manual intervention.

You will determine the necessary number of servers that you deploy in each cluster based on two key factors;

1. To provide high availability you require a minimum of two servers per cluster but it is common to choose a minimum of three servers so that the failure of any single node still leaves two servers available to process the necessary traffic, which avoids overloading the single remaining server
2. To successfully handle the single-node-failure scenario you must ensure that under normal conditions each node has sufficient spare capacity to handle a share of traffic from the failed node(s). For example, in a 3-node deployment each node must normally run at less than 66% CPU (or perhaps 60% CPU to be on the safe side) so that the remaining two nodes can pick up their share of the additional traffic without running out of capacity

For the Management and Developer Portal clusters these two rules will typically always lead you to choose a cluster of three servers since the overhead from users accessing the user interfaces via their browser is not generally sufficient to result in heavy load on the servers. The Gateway service however needs careful sizing to be able to meet the traffic demands of your deployed APIs, and very high API call rates may also require some additional capacity in the Management service to handle the processing of analytics events that are generated by the Gateway. More details on sizing each of the clusters can be found in Section 5.

Deploying multiple nodes within a cluster provides good protection against the failure of an individual server process but we also need to consider other failure scenarios that may occur within a given region. For example, in a VMWare ESXi deployment each of the servers are a virtual machine running inside the ESXi server, so a failure of that ESXi server will affect all the servers in the cluster. To mitigate this scenario customers often choose to use multiple ESXi servers which could be in different subnets, pods or availability zones within the region to reduce the impact of other failures like network interfaces etc. With each of these failure scenarios it is important to identify how the deployment would react to the outage and plan the topology so that it has sufficient resilience to handle the failure.

The configuration of local load balancing (within the region/ data centre) has slightly different requirements depending on which of the components are being managed, the details of which are also discussed in Section 5. The two key scenarios are stateless requests like API calls being routed to the Gateway service, versus session-based web browser calls to the API Manager / Cloud Manager or Developer Portal.

4.3 External and Internal API Exposure

Some API Connect customers have API programmes which include both external/partner interactions as well as APIs served for use entirely within the enterprise which leads to a question of how to structure the logical and physical structure of the deployment to meet those two use cases;

- There is commonly a significant separation between the set of APIs made available to the externally facing subscribers and those of the internally facing subscribers or the preferred mechanism for authentication those application developers to the Developer Portal, so often a separate Catalog is created to service each scenario
- These Catalogs might be within a single Provider Organization if there is overlap between the set of APIs published to each group of consumers, or the Catalogs might be in different Provider Organizations; the latter being common if there are distinct groups of people responsible for managing the APIs in each case
- Regardless of the logical layout there is still the question of how much sharing of infrastructure there should be between the two cases

The thought process in choosing the appropriate topology for this scenario is similar to the discussion in section 2.5 on Environment separation in that it is important to understand that whenever there is sharing of infrastructure between two use cases there is the potential that a change or an issue in the shared infrastructure will affect both cases at once. For example, it is not recommended to share infrastructure between a public facing Production environment and an internally facing development or test environment.

There is however a potentially valid scenario where both the public facing and internally facing scenarios are the same "class" of environment – typically if they are both Production services. This can clearly be achieved with a single deployment if the internal consumers are made to route their requests out to the public facing network and then back in through the DMZ and load balancer as shown in the following diagram, however this is not generally desirable from a security or performance perspective as it means internal traffic being routed over the public network.

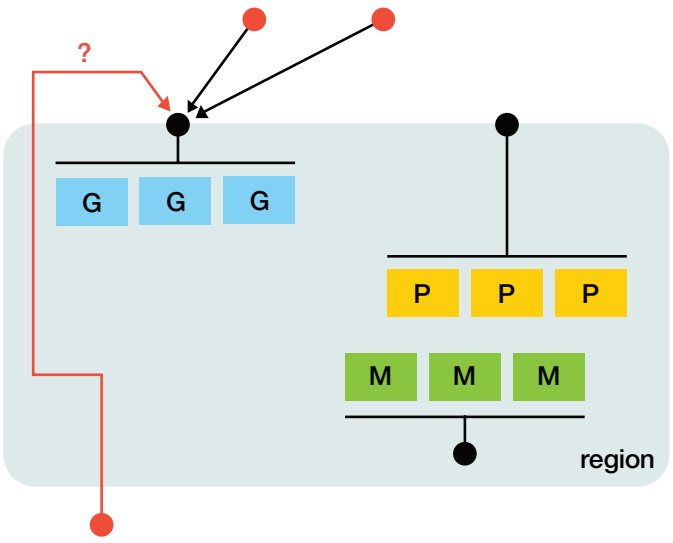


Figure 9: **Forcing internal consumers to follow the same path as external (not recommended)**

To avoid the undesirable security and performance implications an alternate approach is to configure a second Gateway service so that externally facing APIs will be deployed to one cluster, and internally facing APIs deployed to the other as shown in the following diagram;

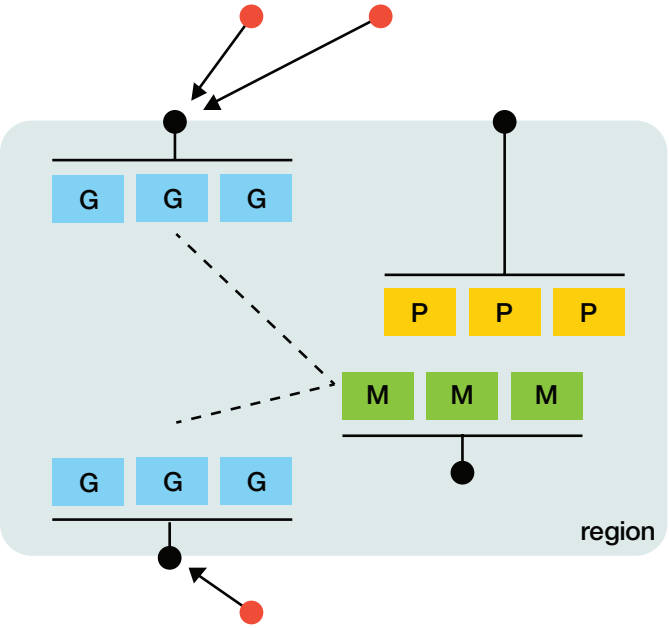


Figure 10: **Separate Gateway service for internal and external traffic**

This approach is used successfully by customers who are comfortable with the following implications of the topology;

- There is a shared Management service and Developer Portal cluster so any problems in those tiers could affect both the externally facing and internally facing users at the same time
- Upgrade is controlled by the (shared) Management service so the external and internally facing Gateway services must be upgraded at the same time, which introduces schedule dependencies between the two use cases that may cause conflict within the enterprise
- Since it has both externally and internally facing components the cloud deployment will span multiple network zones starting with the DMZ and reaching into the protected or private network zones depending on your network configuration. Some customers have network security guidelines which restrict the number of zones a set of connected servers can span which might prohibit this type of deployment

Customers that are not comfortable with the conditions described above arising from the shared Management service can choose to deploy two separate Clouds – one to handle the externally facing scenario and one for the internally facing scenario as shown in the following diagram;

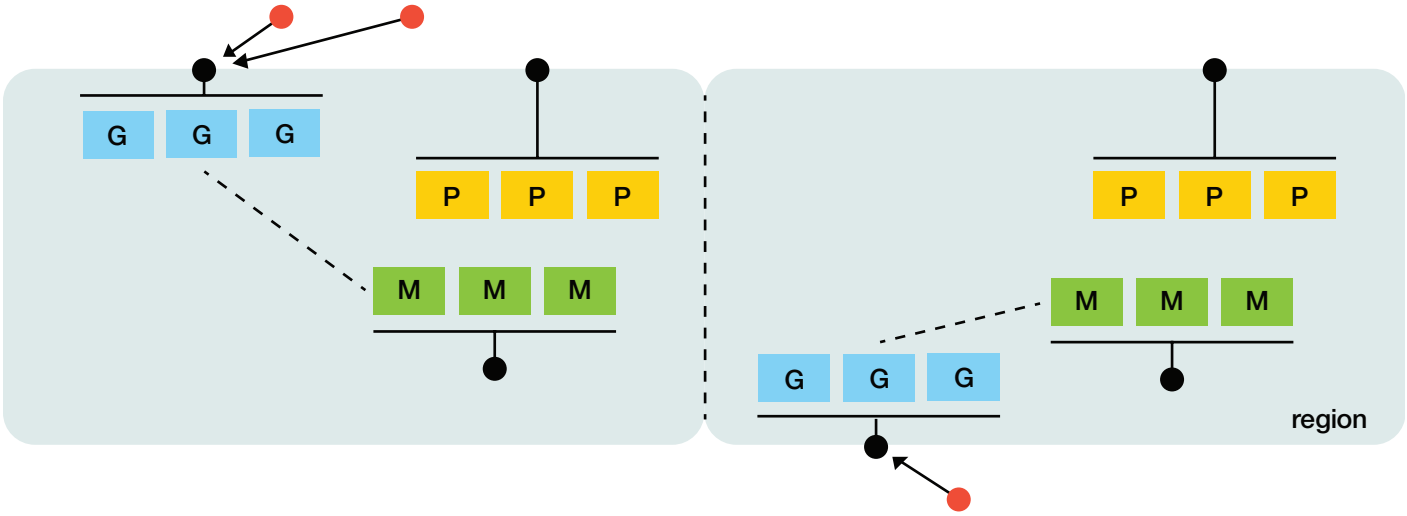


Figure 11: **External and internal traffic using two separate clouds**

4.4 Dual Region with High Availability

For on-premises customers the most common production deployment scenario is to have a single cloud clustered across two regions (data centres) as shown in the following diagram.

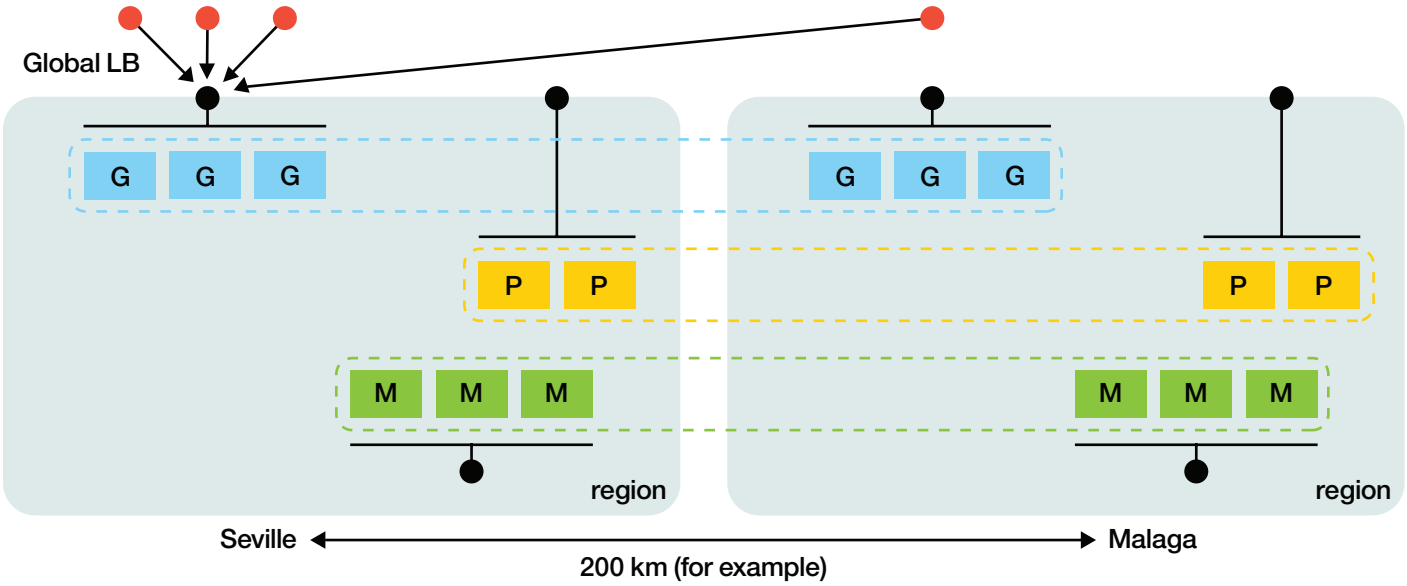


Figure 12: **Dual regions with high availability**

As shown in the diagram this approach provides two levels of resilience – firstly there are two regions so there is isolation from a total outage of a region, but also there are still multiple servers in each region for a given component so there is high availability within the region as well. The latter means that for the simple "single server" failure scenarios both regions will continue to function as normal.

In exchange for the additional resilience that comes from running in two locations there are some new areas that need to be considered in the deployment;

1. For load balancing we must now consider not only "local" load balancing (within the region/site), but also "global" load balancing that allocates traffic between the regions. Global load balancing typically takes the form of either simple DNS configuration, or using a managed DNS provider such as Dyn or Amazon Route 53

2. Depending on the type of traffic it may be appropriate either to spray traffic evenly across both regions, or in some cases customers may choose to route all traffic of a given type to a single "primary" region to achieve certain other goals resulting in a hot standby style configuration. More discussion of the reasons why that might be desirable can be found in section 5.
   - Note that even if user traffic is being routed in a hot standby fashion to a single region the API Connect deployment infrastructure itself is always active/active to ensure the necessary data replication takes place to allow the second region to immediately serve the same traffic workload if the primary region becomes unavailable
   - As such there is no "standby" server configuration for high availability in API Connect — that scenario is better described as "disaster recovery" and is achieved by restoring a backup into a new set of deployment infrastructure as discussed in section 6.1

3. Since the deployment now spans two distinct geographic locations there is an increased risk that the network connectivity between those locations will be less reliable than within a single location – for example higher latency/response time, packet loss, or the two locations to become disconnected from each other for a period. Each of those cases can affect the replication of state between the two regions and so increases the set of failure scenarios that should need to consider in testing your deployment.

It is important to note that to successfully handle the case where a single region is completely unavailable you must size both regions so that they can each handle the total traffic workload of your deployment – for example if API call traffic is generally sprayed evenly across two regions then each region must be running at less than 50% utilisation so that one region has capacity to pick up the traffic from the other region in the event of a failure.

Following the line of questions in section 3.3 an additional point for consideration is that you may have decided that not all functions have to be present in all regions. If so, you will have made a conscious decision that if the "wrong" region becomes unavailable then you may not be able to carry out certain actions – for example deploying new APIs.

4.5 Bluemix Dedicated Deployment

IBM Bluemix Dedicated is a hosted Bluemix environment that is dedicated to your sole use and so provides enhanced security, privacy and performance when compared to Bluemix Public. The functionality provided by API Connect in Bluemix is largely identical to the on-premises deployment but the administration and operational management of the deployment is carried out by IBM rather than you as the customer – this means that you and your team can focus on the higher-level tasks of defining and managing your APIs without having to commit time and resource to the deployment and day-to-day management activities.

As the IBM team handle the administration and operational management of your Bluemix Dedicated deployment the low-level details are not discussed in detail in this whitepaper – the infrastructure will be put in place by the IBM team on your behalf following similar guidelines as discussed here, but adapted for the specifics of the Bluemix deployment infrastructure. The following sub-sections describe the Bluemix Dedicated topologies at a high level to give the background you need to understand the deployment options as you engage with the IBM team.

For more information on Bluemix Dedicated in general please see the following site;
ibm.com/cloud-computing/bluemix/dedicated

4.5.1 Single Environment Bluemix Dedicated

A single Bluemix Dedicated environment (the default option for Bluemix Dedicated) is analogous to the "single region with high availability" topology described in section 4.2 in that it provides resilience within the region but not to failures of the whole region. As the customer, you can choose which data centre your Bluemix Dedicated deployment is placed into from the list of 25 of more IBM Cloud data centres so you can align the deployment with the needs of your consumers, your on-premises data centre location or other factors.

Since the administration of the deployment is handled by IBM you will be consuming the offering "as a service" and you will experience the deployment as a single logical service as shown in the diagram below. Internally IBM will carry out an appropriate deployment to provide high availability within the region using a default pattern which forms the starting point for all Bluemix Dedicated deployments. The size of your specific deployment might then be determined based on information you provide about the expected volume of traffic, style of usage etc. That information will come in part from the predicted API call volume that you purchase but will also typically involve further technical conversation between IBM and the customer, for example to discuss your typical traffic patterns/peaks and expected rate of API call concurrency, which are not reflected directly in the purchasing metrics.

As shown in the diagram below one of the important aspects of Bluemix Dedicated is that there is the option to have a private network tunnel to connect the Bluemix Dedicated internal network to your customer on-premises network, typically using either a VPN or DirectLink connection. It is common for the network routing to be configured so that the API Manager interface is only accessible through this VPN connection

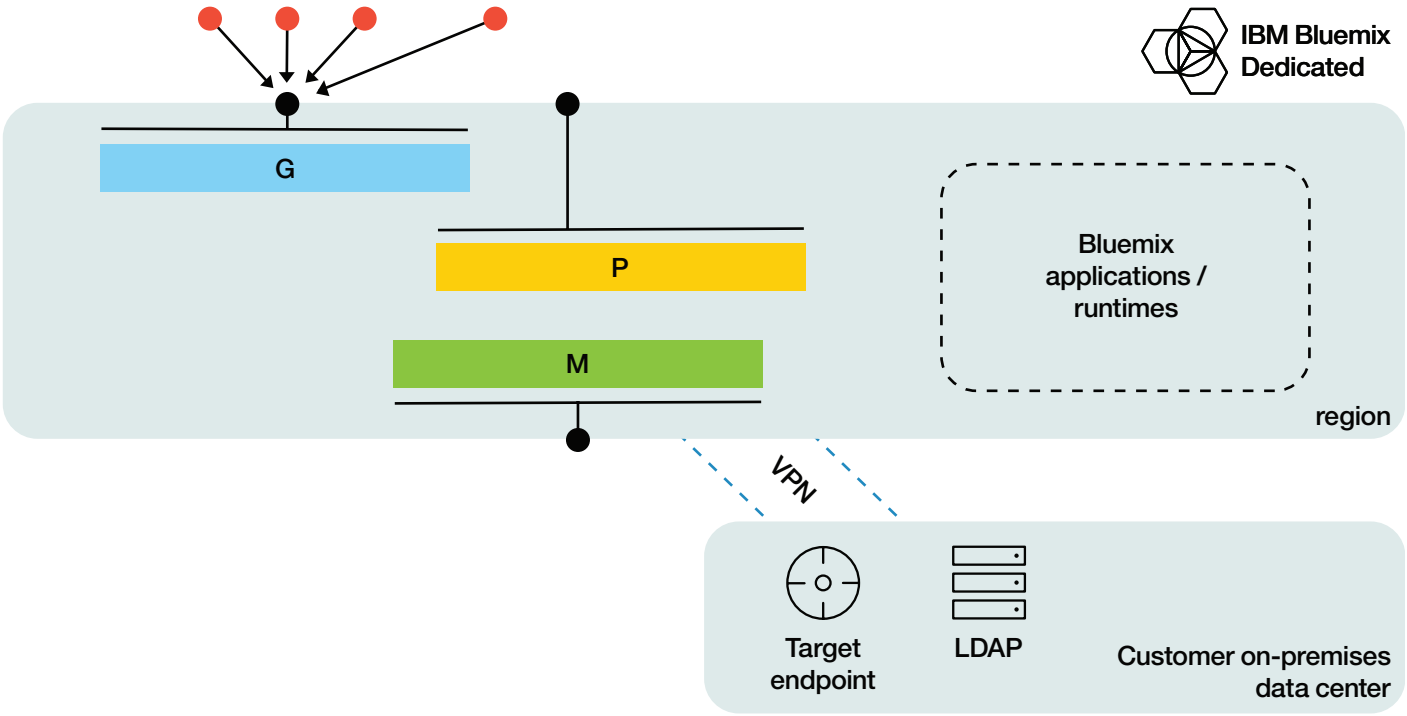


Figure 13: **Single environment Bluemix Dedicated deployment**

since the users of that interface are typically a restricted set of people within your organization. Similarly, it is common for the backend services that are being managed by API Connect to be accessed over this private network link when APIs are invoked to protect that data flow. By contrast the Gateway and Developer Portal interfaces are often configured to be accessible over the public internet, for example so that they are available to your partners and deployed applications in the wild. Each Bluemix Dedicated customer has slightly different requirements so the IBM team will work with you to determine what is appropriate in your case.

IBM also configures appropriate local load balancing for the deployment so that you can access the API Manager, Gateway, and Developer Portal endpoints without needing to be aware of the exact server topology that has been put in place on your behalf.

The net result for Bluemix Dedicated is that you can avoid spending your time and resource handling the system administration aspects of the solution and focus on the creation, deployment and management of your business APIs.

4.5.2 Two Environment Bluemix Dedicated

To provide higher availability and resilience IBM provides the option of a two environment Bluemix Dedicated deployment as shown in the following diagram. Like the dual region topology described in section 4.4 this topology ensures that your API Connect infrastructure can continue functioning even in the event of an outage of one of the regions.

The same topics and questions are relevant in considering this topology as for the on-premises deployment case – for example how you configure the global load balancing capability for the deployment and whether to route traffic actively to both regions or just one of them depending on the type of workload in question.

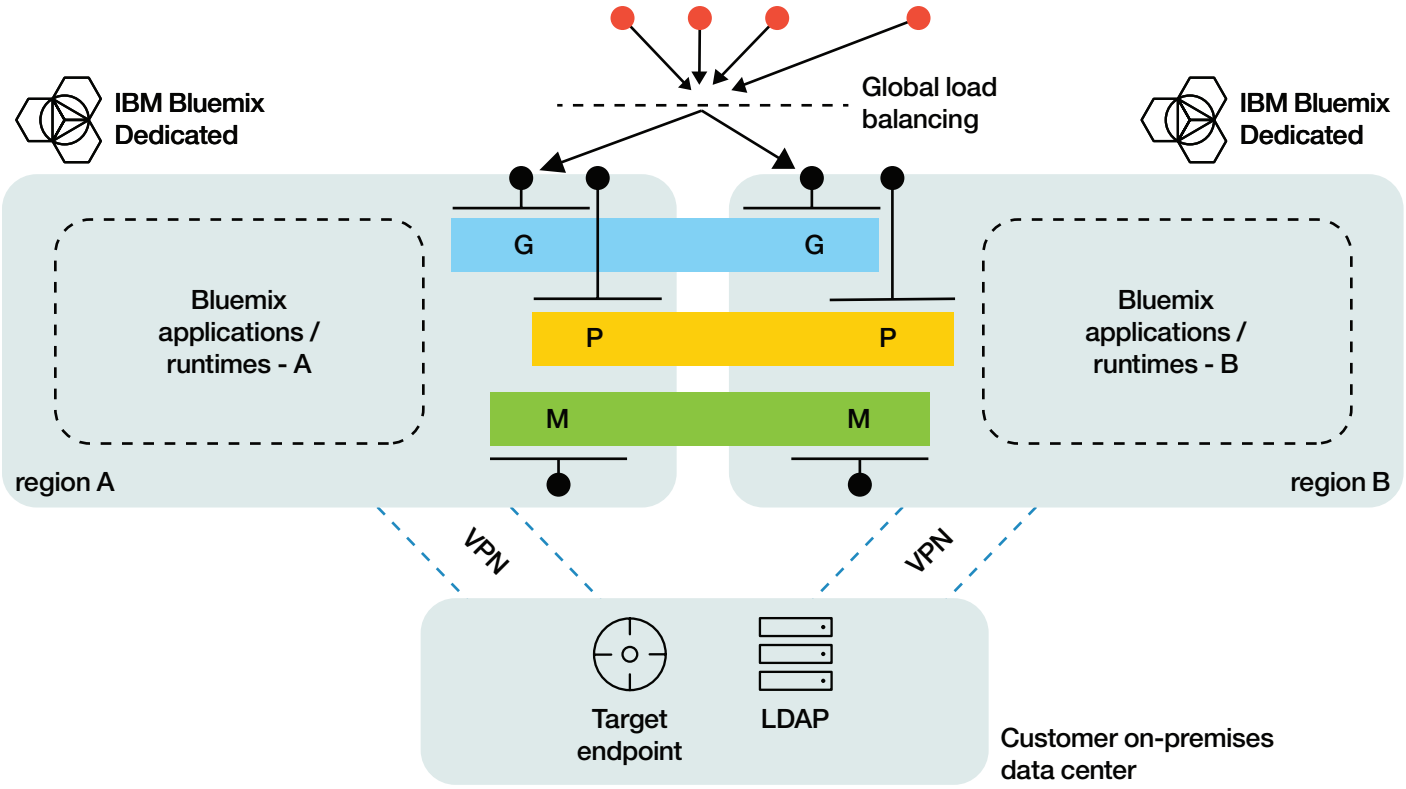


Figure 14: **Two environment Bluemix Dedicated**

As administrators of the deployment the IBM team have a recommended pattern of deployment that they will discuss with you which builds on their experience of managing API Connect deployments on behalf of other similar customers and they will discuss your requirements in the context of this recommended pattern.

An important note for two-environment Bluemix Dedicated is that the API Connect deployment is configured to form a single logical cloud clustered across both regions so that each location can serve requests identically and you can fail over from one to the other without interruption in traffic. By contrast the Bluemix applications and runtimes are two independent (not clustered) pieces of infrastructure so any applications you deploy - including Loopback applications as part of API Connect – must be deployed to both locations separately to provide the same application functionality in both regions.

The two-environment Bluemix Dedicated topology is most commonly used for providing higher availability within a given geographical area where the two deployment regions are relatively close together – for example Amsterdam and Paris in Europe. This ensures that the consumers of the deployed APIs have relatively consistent latency in communicating with each deployment location.

4.6 Hybrid Gateway Topology

There is an emerging trend for some customers to pursue a hybrid deployment topology where certain components of the API Connect deployment are separated into different locations. This is often part of an enterprise's progression from a traditional on-premises-only infrastructure towards a goal of running more systems "in the cloud". In this context hybrid topologies provide useful stepping stones in gradually moving from one approach to the other.

The most common hybrid topology we see customers asking for is the "hybrid gateway" topology shown in the following diagram, in which a customer wants to have the Gateway endpoint deployed in their on-premises network. In this case the Management and Developer Portal endpoints are deployed in Bluemix but the API call traffic is routed directly to the customer's data centre;

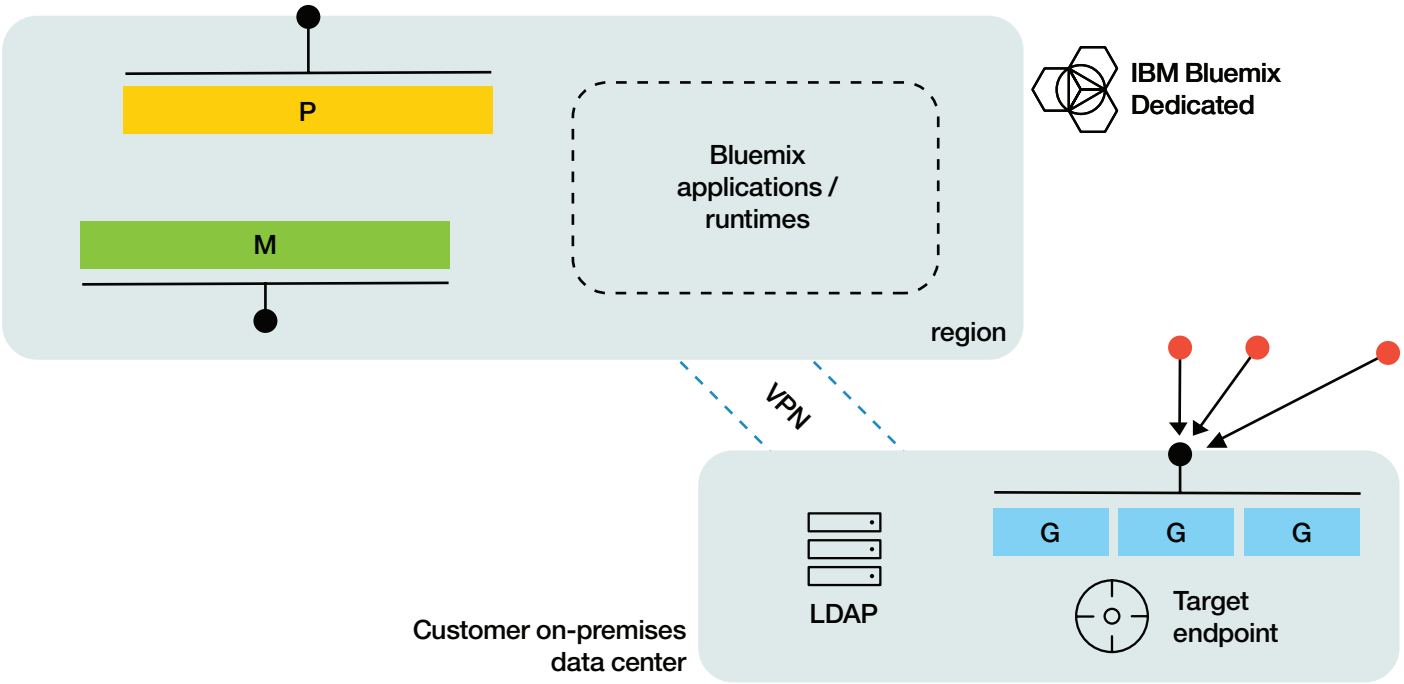


Figure 15: **Hybrid gateway topology**

There are several different reasons why this scenario is of interest to customers;

1. The customer already has an on-premises DataPower gateway deployment which exposes endpoints publicly through the DMZ and they want to use API Connect to deploy APIs to that existing infrastructure, but don't want to be responsible for managing the "extra" API Connect infrastructure (e.g. Management and Developer Portal services) themselves
2. Highly performance or availability-critical scenarios where the customer wants to place the Gateway in the same immediate network location as the target endpoints to optimise the API call path and avoid traversing the private network connection between the Bluemix Dedicated region and the on-premises data centre

Separating the Gateway component from the other parts of API Connect as in this hybrid gateway topology introduces some additional considerations for this style of deployment as described below – in most cases these aspects are acceptable when compared to the benefits the topology brings but it is important to be aware of the implications when making your decision;

- Shared administration of the Gateway instances
  - The on-premises Gateway instances are administered primarily by the customer since they run in the on-premises network, but API Connect requires the ability to configure some aspects of those instances and deploy API definitions so that they can be acted on
  - This means that both the customer administrators and IBM have some level of shared administration responsibility for the infrastructure, and can increase complexity when updates are required to the configuration or there are issues that must be diagnosed, either because it may not initially be clear which party is responsible or because it requires coordination between the two organizations

- Transmission of Analytics events
  - As described in section 2.1 the Management service is responsible for storing the API event analytics data that is generated by the Gateway so that it can be queried later
  - In this hybrid gateway topology the analytics data flows from the on-premises Gateway to the Management service in the cloud via the private network link which means that there is an increase in network latency and risk of interruption in connectivity for the data flow compared to the Gateway running adjacent to the Management service
  - This means that for high API call traffic rates it is important to size the bandwidth provided by the private network link, and there is an increased risk that API event data may be lost if the Gateway is not able to transfer the data to the Management service fast enough to keep up with the incoming traffic

4.7 Global Deployment for Geographical Affinity

One of the most advanced topologies is a global deployment in which multiple regions are spread across continents to provide a single logical cloud. This is typically driven by scenarios in which the same set of APIs are to be exposed to one or more applications for which client requests originate from anywhere in the world, and there is a strong desire to minimize response time by ensuring that the API request is served entirely within the closest deployment region to the caller.

As with all multi-region deployments the global load balancing configuration is very important in routing traffic to the correct deployment region. In the diagram shown below the API call traffic has global load balancing configured so that the request is routed to the geographically closest access point for the caller – for example application requests from users in Europe are served by the European deployment region.

What is often overlooked when first considering this scenario is the location of the backend services that the Gateway is forwarding requests to. Ideally the backend service endpoints (and all their dependencies) should also be replicated in all deployment regions to minimize latency for the requests and ensure availability of the end-to-end API call; there is reduced benefit from having Gateway instances in each geography if there is only a single instance of the backend service in an individual geography since if that backend service cannot be reached then all API calls will fail. One key scenario that may mitigate that impact is if API responses can be cached locally by the Gateway so that most API calls never need to contact the target service.

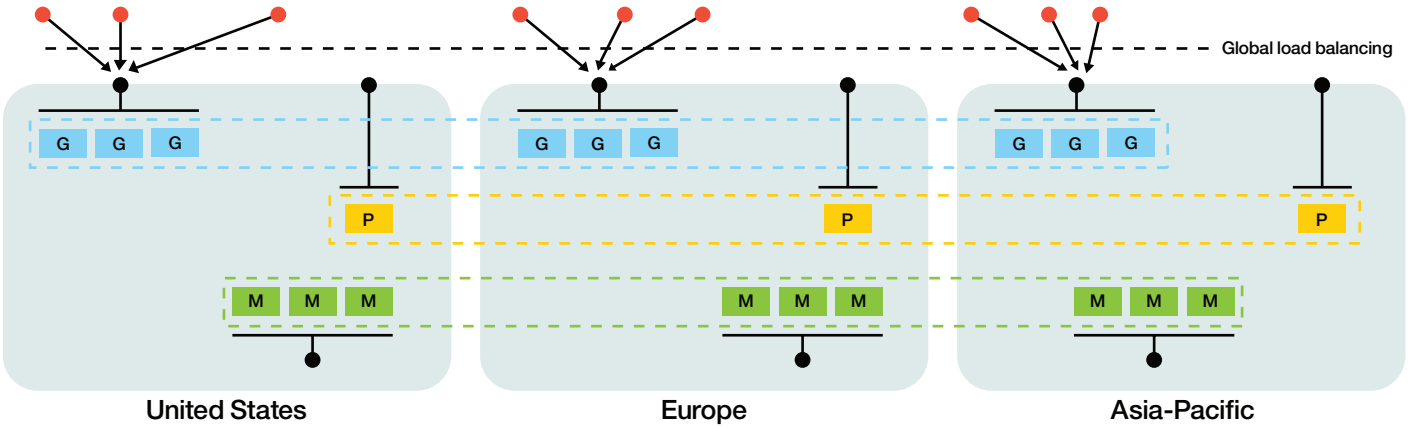


Figure 16: **Global deployment for geographical affinity**

4.7.1 Reduced Topology Global Deployment

A global deployment with three (or more) regions increases the risk of network partitioning and the potential for resulting replication problems so it is especially important to consider whether all the deployment components need to be deployed in all regions. Instead of the fully replicated topology shown in **Figure 16** in some cases you might choose a reduced topology like that shown below if certain requirements are met, for example;

- All the application developers are internal, so less reliance on the Developer Portal availability
- API developers / administrators are mostly internal and/or in a single geography, so less need to replicate the Management service

This creates a "satellite gateway" region that only has the Gateway instances deployed (and hopefully replicated copies of the target service endpoints locally as well as mentioned above), and reduces the extent of the Management and Developer Portal services. The satellite gateway approach does however have the same implications as the hybrid gateway case in section 4.6 because the Management service is always outside the local region.

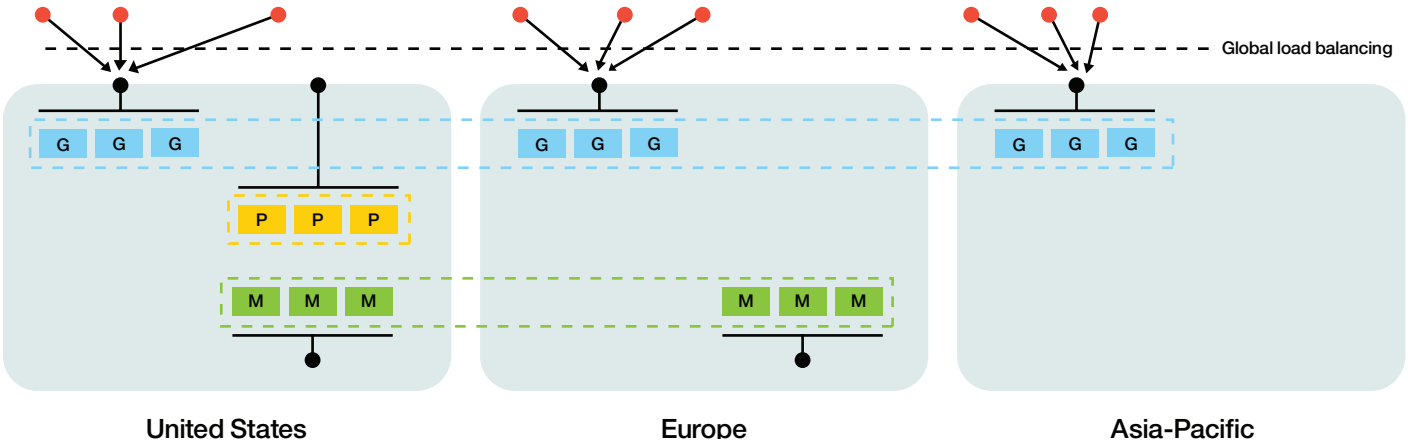


Figure 17: **Reduced replication global deployment**

5 Key Points for Multi-Region Deployments

In this section, we discuss each of the major deployment components in turn and highlight the key points of interest when putting in place your deployment topology and the infrastructure that surrounds it.

5.1 Deployment Components

You will hopefully remember the following component overview diagram from section 2.1 where we introduced the major components of the API Connect deployment.

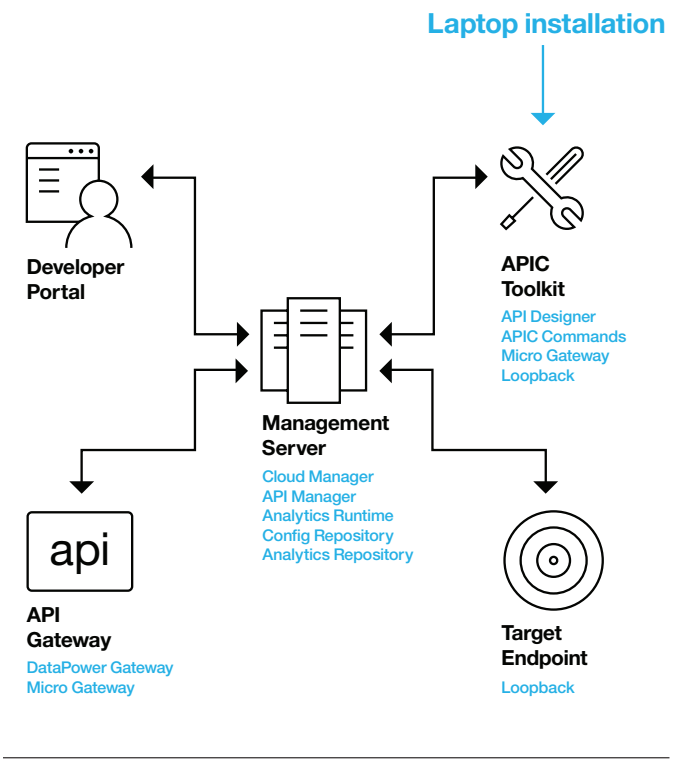


Figure 18: **Deployment components**

The following sub-sections will look at each component in turn and deep dive inside the component where relevant to discuss the details of the way it works and the requirements it has in relation to the deployment topology. Each sub-section starts with the following "at a glance" table that allows you to get an immediate overview of the key points, and then goes in to more details on the background and reasons behind those statements.

Session affinity	
Load balancing	
Persistence	
Replication	
Failover behaviour	
Deployment location	
Cluster sizing	

Table 3: **At a glance: Example**

5.2 API Gateway

Session affinity	None – requests are stateless
Load balancing	Round-robin distribution for both global and local load balancing
Persistence	Only for domain configuration that provides the framework for serving APIs Rate limit data is stored in-memory only – no persistence
Replication	Rate limit information is replicated between peers using IP multicast
Failover behaviour	No failover required (independent instances) unless using AO for front-side load balancing
Deployment location	Suitable for DMZ deployment if desired
Cluster sizing	Varies in line with API traffic rate

Table 4: **At a glance: API Gateway**

API call processing is typically the most important component of the system from an availability and resilience perspective and is designed to be largely stateless (and not require **session affinity**) so that the system can be scaled efficiently and maximise performance. To support this goal, well defined customer APIs are also implemented to be stateless in nature from the perspective of the Gateway tier, so that any instance can be selected to process the incoming request.

Since the Gateway processing itself is stateless the **load balancing configuration** can be configured to a stateless technique as well, and we recommend using "round robin" for local load balancing in order to evenly spray traffic across the available nodes within the region. Some customers also consider a "least connections" or "fastest response time" policy for local load balancing, however this can lead to unexpected behaviour in failure scenarios since in many cases an error response will be returned immediately (eg couldn't access the backend service) which may cause the load balancer to route more and more traffic to the failing node because it observes faster response times.

The global load balancing policy (for selecting a region) will following one of two patterns depending on your preference of whether both regions actively serve traffic;

1. Round-robin approach for global load balancing - for example a simple DNS mapping that configures the external endpoints for both regions, and allocates each address at random to clients, or a more intelligent DNS allocation based on health-check knowledge of the endpoints
2. Routing all traffic to a "primary" data centre – even though the Gateway tier can serve traffic statelessly in both locations some customers choose to route traffic to a primary site under normal conditions, perhaps to have more efficient connectivity to backend systems, or simplify the steady-state topology

Note on simple DNS configuration:

If you choose to use simple DNS configuration with multiple service endpoints for your global load balancing policy it is important to consider the time it will take for a client application to pick up any changes you make to that configuration. For example, if region 1 fails then any client applications that were given the region 1 endpoint from DNS will continue to use that endpoint until they next refresh from the DNS server, even though all the requests may be failing. This means that you should configure the DNS time to live (TTL) to a short time interval such as 1 or 5 minutes to instruct the client app to refresh its DNS selection regularly – the TTL will dictate the duration of time for which clients may experience failing requests in the event of total failure of a region.

It is also important to test regularly that the client applications do in fact honour the DNS TTL setting otherwise some clients may never pick up the DNS changes you make and all their requests will be failing, even though the other location is successfully serving traffic.

Although multi-endpoint DNS configuration is cost effective and easy to configure, the failure case described above often mean that customers will look at more powerful (and thus more expensive / less simple to configure) solutions to minimize the client applications' experience of outage scenarios.

Persistence and replication of data within the Gateway falls into one of the following categories;

- API Connect domain configuration — each instance has a local copy persisted to disk. No replication to other instances is required
- API configuration data — each instance has a local in memory copy of the data. No replication to other instances required. Must be reloaded from the Management tier if a DataPower instance is restarted, during which time that instance will not be able to successfully serve API traffic
- Rate limit state data — stored in-memory and replicated between instances using IP Multicast with a replication interval of the order of 10ms. See below for more discussion
- OAuth token data is self-contained within the token so does not need to be replicated or persisted across the cluster, as long as all the instances are using the same cryptographic configuration data — which will be configured automatically by API Connect

Since replication of rate limit data (SLM peering in DataPower terms) is carried out over IP Multicast the observed behaviour for multi-region deployments is sensitive to the network configuration that is put in place. In most networks IP Multicast packets are restricted to transmission within the local subnet so a gateway service which spans two regions (typically two or more subnets) will only be able to replicate data within each region. If round-robin global load balancing has been configured across both regions this will mean that clients for a single application (clientID) might be able to invoke twice the number of API calls that is specified by the rate limit – one quota for each region. If rate limit enforcement is particularly important to your scenario you may choose to mitigate this behaviour by either routing all API traffic to a single active data centre, or by configuring your network to allow the multicast packets to propagate between the two (or more) regions.

As each individual instance can serve traffic independently of the others there is no specific failover required if one of the instances fails – the other instances will happily accept their additional share of the traffic. In the event of a **failure** there may be a small window of rate limit data which may not have been replicated, but this is typically only a few milliseconds so does not represent a major concern to most customers.

There is one main scenario where **failover** may be relevant to the Gateway which is if you are using the Application Optimisation (AO) feature to provide self-load balancing of incoming requests across the Gateway cluster instead of using an external load balancer. In that scenario one of the Gateway instances will be the current owner of the virtual IP address that represents the cluster, and if that instance fails the cluster must elect a new instance to become the owner of the IP address. The AO takeover process uses gratuitous ARP, which is typically configured with a granularity of around 10 seconds for the failure to be detected and another instance to begin serving new connections.

DataPower is a component with a long heritage of security and hardening and as such is well suited for a **deployment location** in the demilitarized zone (DMZ) for externally facing scenarios. Some customers have guidelines which require use of the physical DataPower appliance form factor when placed in the DMZ but increasingly customers are also comfortable with deploying the virtual appliance option in the DMZ as well. For internally facing scenarios the DataPower gateway can also be used (in any form factor), but some customers also consider the MicroGateway, particularly if they want to assign ownership of the Gateway infrastructure to the same that owns the set of APIs or microservices that it is fronting.

The **cluster sizing** decision for the API Gateway is based on the volume of the API call traffic. As discussed in section 4.2 you will need at least two or three instances to provide resilience to failures of individual instances, but you still need to determine whether to deploy additional nodes beyond that number to serve your API traffic. The exact capacity of a Gateway instance is very dependent on the style and complexity of APIs that you deploy there – for example simple APIs that are effectively just a proxy to an existing backend service are faster and more efficient to execute than if you are aggregating responses from multiple backend services or doing protocol transformation from SOAP to REST. Your performance profile will also depend on how much CPU, memory and disk you have allocated to your Gateway instances, and whether they are physical or virtual instances, which makes giving generic guidance inaccurate or misleading. Our recommendation is to engage your local IBM technical sales contact to facilitate a discussion about your specific APIs / deployment topology which they can use to help provide guidance on the recommended instance size and cluster size.

5.2.1 Resilience to Failure of Other Tiers

The Gateway tier interacts with the Management service to both consume metadata about the deployed APIs/Plans, and to send API event data to the Management service's analytics store.

The Gateway makes a series of poll requests to the Management service approximately every 20-30 minutes to refresh its view of the APIs that are deployed and the application subscriptions etc. It has a full list of Management servers in the cluster and will pick one that can be contacted, but even if all the Management servers are unavailable the API traffic will continue to be served successfully based on the latest locally stored view of the configuration state. Note that this was not always the case so if you are running a version of API Connect earlier than v5.0.7.0 and DataPower firmware 7.5.2.4 then API calls may begin to fail after a period of time if the Management service cannot be contacted.

The Gateway also generates API event analytics records that log metadata about the API invocations that are being processed such as response time, HTTP response code etc, and may also log the payload of the request and response body if the "Activity Log" policy has been configured in the API assembly. In order that the client's API request be completed as quickly as possible that API event record is sent to the Management service asynchronously (on a separate thread) and may also be batched with other API event records if the API traffic rate is sufficiently high. The records are stored in an in-memory buffer before they are transmitted, so if all the Management servers are unavailable the data will be retained in memory and retried a little later, however there is the possibility that if the duration of the Management service outage is long enough, or the API traffic rate is high enough then the in-memory buffer will become full and some analytics data may have to be discarded. There is also a maximum data size on the memory buffer so it will be used up faster if you have payload logging enabled. Note that mainline processing of API requests is not affected if the memory buffer becomes full – only that the analytics records for those calls may be lost. In scenarios where the historical analytics data is not being actively used this does not generally present a problem, but if you are billing your application consumers directly based on the specific number of API calls they make then this potential loss of data represents a window of lost revenue because of the Management service outage.

5.3 User Interface Traffic

Session affinity	Replication of user HTTP Session state exists for API Manager, Cloud Manager and Developer Portal for failover, but recommended to route back to same instance under normal conditions
Load balancing	Support recommendation to route back to same instance by configuring both global and local load balancing for session affinity-based load balancing (eg sticky sessions or similar)
Persistence	HTTP session replication is in-memory only
Replication	HTTP session replication takes places over TCP/IP
Failover behaviour	N/A
Deployment location	N/A
Cluster sizing	N/A

Table 5: **At a glance: User interface traffic**

This section discusses the aspects related to user interface traffic handling in API Connect. From a deployment component perspective, this covers two separate areas - with the API Manager and Cloud Manager user interfaces being served by the Management service, and the Developer Portal user interface being served (unsurprisingly!) by the Developer Portal cluster, however the requirements and recommendations for these two cases are largely the same so we have combined them into a single section here.

The "sameness" of the two scenarios stems from the fact that user interface sessions take place under an HTTP session that is established with the relevant server infrastructure and so the default assumption is that the networking infrastructure will maintain **session affinity** and ensure that all requests for that session be routed back to the same server instance that served the original request.

In fact, API Connect implements HTTP session replication out of the box for each of the three user interfaces so it is possible to have subsequent requests routed to any node in the cluster, however in practice there is a small chance of some race conditions causing errors due to replication of data around the cluster if you were to implement a true stateless/round robin **load balancing** approach;

- The HTTP session may not be replicated immediately to other nodes in the cluster so there is the potential for "not authorized" errors to occur when the user first starts up the UI and establishes their session
- Read requests might be served by a "local" database read-cache so there is the potential for write actions that have just been made to not be reflected immediately in read requests

As a result, we still recommend to maintain the session affinity model for load balancing under normal conditions, and fall back to an alternate server only in the error cases where the initial server may have been taken offline or had a failure.

At a technical level, the HTTP session **replication** capability in both the Management service and the Developer Portal use a standard TCP/IP mechanism for replicating the necessary data between the servers in the cluster so there is no extra work required to maintain that replication if the cluster is deployed across multiple regions or subnets (in contrast to if it were replicated using IP Multicast).

The replicated session data is stored in-memory only and is not **persisted** to disk. This means that if you were to stop the entire Management service (all servers in the cluster) and then bring it back up again the user would have to create a new HTTP session. A total copy of the session data is replicated (in memory) to all servers in the cluster so if there are one or more Management servers active then the session data will be retained, including re-replicating it to other servers once they come online. As a result, the only time that session data would be lost (and the user have to create a new HTTP session) is during an upgrade scenario or a total infrastructure outage, which is not typically a major concern.

5.4 Management Tier

Session affinity	As described for UI traffic in section 5.3
Load balancing	As described for UI traffic in section 5.3
Persistence	Configuration data (APIs, Products, Apps, Subscriptions etc) in a persistent database API call and audit analytics data stored in integrated ElasticSearch instance
Replication	Persistent database uses a primary + multiple secondary model, with fully copy replication ElasticSearch is a sharded replicated data store (partial copy on each node, where suitable)
Failover behaviour	For the persistent DB, one of the "secondary" Management servers is responsible for electing the new primary. Secondary servers are also able to serve read requests even when not connected to the primary. ElasticSearch is a peer-based model so remaining nodes continue to function, but data from the failed node may need to be redistributed to the remaining instances.
Deployment location	Private or protected zone. Not recommended for deployment in the DMZ
Cluster sizing	At least two (or three) nodes in each region to achieve high availability. UI or CLI-based traffic load not generally sufficient to require additional nodes. High API call rate or payload logging scenarios may drive need for additional nodes for analytics storage.

Table 6: **At a glance: Management tier**

As mentioned in section 2.1.1 there are two main functions provided by the Management service which determine the characteristics of the cluster from a deployment and failover perspective;

- Configuration database which stores the details of APIs that you have deployed to a Catalog, application developers that have registered as a developer organization, their Applications and Subscriptions etc
- Analytics repository (ElasticSearch) that stores the historical record of API events that have been processed by the API Gateway and can be retrospectively queried via the API Manager interface

These functions, and the server in which they reside are not security hardened to the level that would needed for the instance to be deployed in an internet facing zone, so the servers in the Management service should be deployed in a private or protected **deployment location**, and not in the DMZ.

The following subsections discuss each of the main functions in turn;

5.4.1 Configuration Database

The configuration database stores all the data about the objects that you have configured in the Cloud Manager and the API Manager, for example the server instance details, the definitions of the APIs that you have deployed to your Catalogs, the user memberships and role allocations. It also stores the authoritative view of the "Community" information that is set up through the Developer Portal – for example the list of developer organizations, the users contained in those developer organizations, the Applications that they have registered and the Subscriptions they have made to associate their Application with Products that have been published.

Inside the Management servers, API Connect uses an Informix database to provide the **persistence** capability – the database is self-managing and as a user of API Connect you will never interact with it directly – all access takes place through the web browser or REST API interfaces that are provided by the offering.

The database uses a "primary + multiple secondary" full-copy **replication** model where there is a single primary server in the cluster, which is the only server that can process requests to write/update data to the database. Each server in the cluster receives a full copy of the database and the non-primary (referred to as "RSS", for "remote standalone secondary") servers are all read-capable, so if your request happens to be routed to one of the secondary servers it will be processed by that local server if it is reading data from the database, but if it is a request to write data then the secondary server will forward on the request (transparently to the calling application code) to the primary.

One of the non-primary servers is also nominated as the "active arbitrator" (AA) which means it is responsible for triggering the **failover** process by nominating a new primary if the original one fails or cannot be contacted. For multi-region deployments, this style of leader election can sometimes result in the creation of an additional primary server – described as split-brain or cloud disassociation, which led to the inclusion of the "main site" feature which is described in the next subsection.

The general access pattern for the configuration database is typically quite lightweight in that both read and write operations are typically triggered by a human user in either the Cloud Manager or the API Manager user interfaces, or via a CLI or REST API call at the direct request of a human being. This means that it is not generally necessary to have a **cluster size** of more than the standard two or three Management servers that are needed to provide high availability in each region. The load on the Management service will be higher if you are putting in place a continuous delivery pipeline that triggers the deployment or subscription of APIs multiple times per hour, however even this is unlikely to load the system to the point that additional nodes would need to be deployed. (Note: there is also an impact on cluster sizing from the analytics processing as described in section 5.4.2)

5.4.1.1 Mitigating Split Brain Scenarios with "Main Site"

As noted in the preceding section, one of the non-primary (RSS) database servers is nominated as the "active arbitrator" (AA), whose role it is to nominate a new server to take over as the primary (P) if the original primary fails or cannot be contacted. In multi-region deployments, this can lead to creation of an additional primary server if the primary server and the active arbitrator happen to land in different regions as shown in the following diagram;

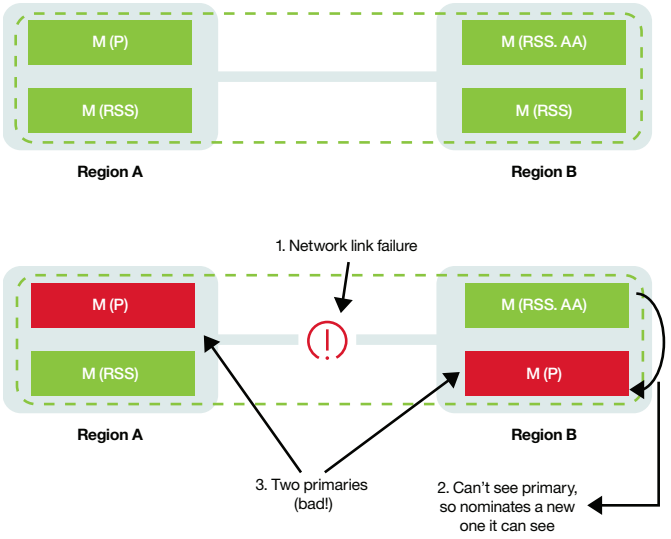


Figure 20: **Split brain scenario (before "main site")**

Once the second primary has been created we are in a state of "cloud dissociation" or split brain in that write activity may be being served by either region, but when the network link is re-established the two sides of the link will not be able to join back together as a single cluster because they have inconsistent views of the data state. Recovering from this situation requires some complicated administrative actions so we want to avoid the potential for this issue to occur.

The "main site" feature is designed to prevent the possibility of a cross-region cloud dissociation by giving the administrator the ability to constrain the set of servers which are eligible to participate in the automatic failover process – effectively limiting which servers can take on the active arbitrator role, and which can become primary. This effectively nominates one of the regions as a "main" location, and disables the "HA" capabilities of Management servers in all the other regions so that they cannot become primary, so that in the event of a network partition between regions there is no opportunity for a second primary to be created, as shown in the following diagram;

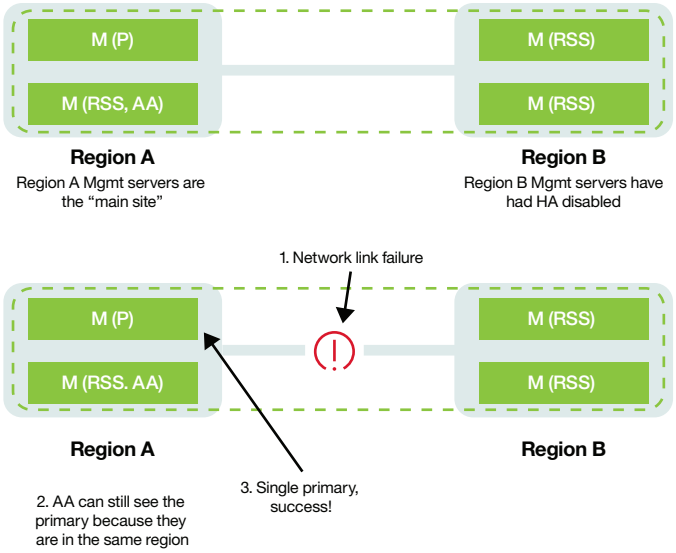


Figure 21: **Network partition, with main site**

Although the main site feature solves the common failure scenario where the network between the two regions fails or is disrupted we have done so at the expense of some reduction in resilience in a less common failure scenario – where the whole of the main site region fails. The second region has had its eligibility to become primary disabled as part of the main site configuration, so it is not permitted to nominate a primary, and there is no way it can know automatically whether the main site region has failed permanently, or whether it will be coming back again soon – it is still able to serve read requests but any write requests will fail because the RSS servers cannot forward them to a primary server.

To resolve this scenario the human administrator must decide that the main site region will not be coming back and is then able to issue some simple administrative commands in the second region to tell those servers that they are now permitted to take on the primary and active arbitrator roles – effectively directing the servers in the second region that they are now the main site as shown in the following diagram;

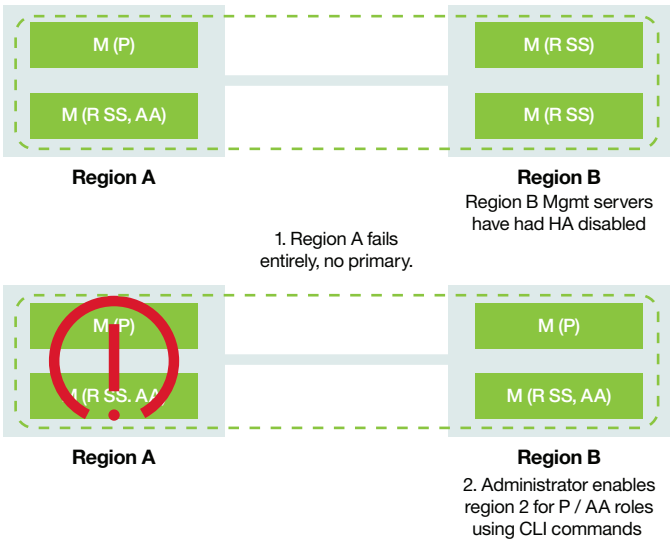


Figure 22: **Administrative re-enablement after region failure**

Once the servers in the second region have negotiated the primary and active arbitrator roles the system returns to normal function and is able to continue serving both read and write requests.

More details about the main site functionality including the specific CLI commands that are executed to enable/disable the main site functionality can be found in the Knowledge Center here; (or search for "main site")
ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/capic_overview_main_site.html

5.4.2 Analytics Repository

The analytics repository stores historical data about the set of API calls that have been processed by the API Gateway as well as audit information about the creation, modification or deletion of APIs or state changes within the API lifecycle. This data is **persisted** and queried using ElasticSearch, which is well suited for this type of write-once/read-many style access pattern.

Replication in ElasticSearch works by allocating each data item (such as a record about an API being invoked) to an "index" and then dividing the contents of each index into a series of "shards" which allows the data contained in the index to be horizontally distributed across the available set of nodes, and for data processing to be carried out concurrently on each shard to improve performance. To provide resilience for the data storage API Connect configures the embedded ElasticSearch index so that as well as a "primary shard" there is also a "replica shard" which stores a copy of the data contained in that shard on a different instance, so that the data will still be available if an individual server is lost un-recoverably.

For high API call throughput scenarios, a large volume of historical analytics data will be generated and stored in your deployment – a rough rule of thumb is approximately 1KB per API call by default. Significantly more data will be stored if you have chosen to enable the "activity-log" policy in

your API implementations as this policy will cause the HTTP request and/or response headers, and/or the API request/response body payloads to be logged, which may be 10s or 100s of KBs per API call depending on your APIs. To store this large volume of data ElasticSearch distributes the shards (both primary and replica shards) across the available set of servers such that each server contains a subset of the shards. This means for a fixed volume of data stored, the more Management servers you add (after two) to your Management service the lower the disk usage will be on each server as shown in the following diagram;

The primary consideration for **cluster sizing** the Management service in relation to analytics is to ensure that the total disk size across the servers is sufficient to store the API event data. API Connect stores the historical analytics data for a period of 90 days so you must consider the total volume of API traffic over that 3-month period when estimating your disk space requirements.

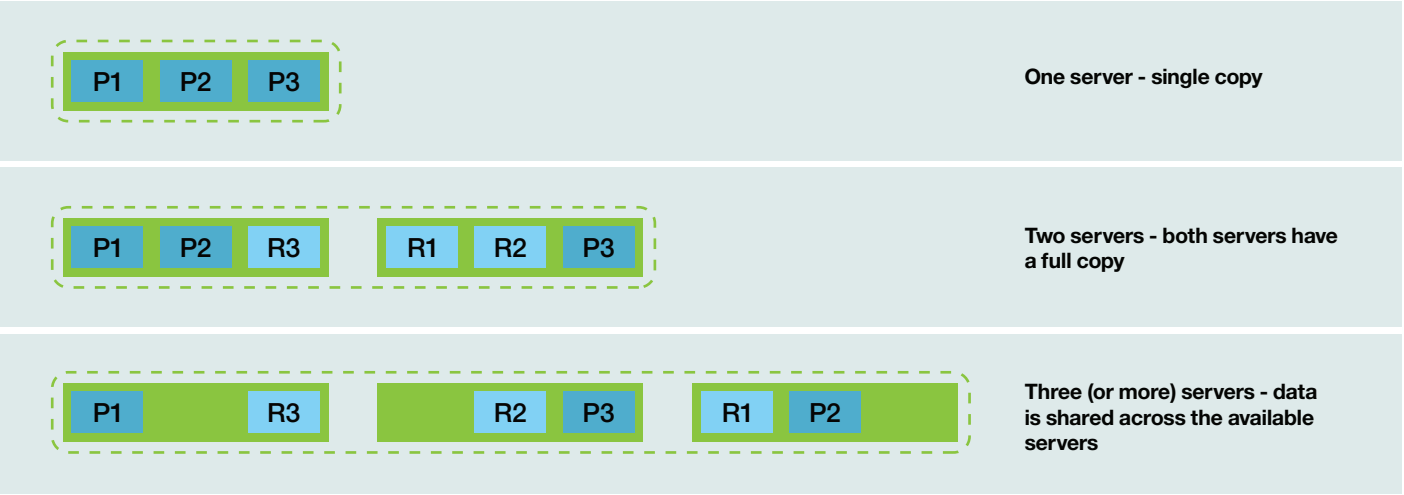


Figure 23: **Example showing ElasticSearch distribution of 3 primary (P) and replica (R) shards across a cluster of varying sizes**

The following give some rough estimates on the number of Management server instances that you will need for various API call volumes, assuming the recommended 300GB disk size for each Management server, and that none of your APIs enable the "activity-log" policy;

Number of Management servers (300GB disk)	Approximate API call traffic rate	Notes
2	100 million API calls/month	Fully copy of data on each server Roughly 1kb data storage per API call 3 months' data stored, so 100GB per month
3	150 million API calls/month	150GB/month data accumulation, so 450GB data every 90 days. Two copies of each item (primary/replica) so 900GB total data storage required
4	200 million API calls/month	
5	250 million API calls/month	

Table 7: **Approximate estimated analytics storage for a given Management cluster size**

The following formula (used to generate the table above) enables you to calculate the number of management nodes that are required to store the analytics data based on the volume of API calls, disk size and estimated event size;

C = Number of API calls/month, in millions (eg 150)

E = Event size, in bytes (eg 1024 for cases where the headers and payload are not logged)

D = Management node disk size, in GB (eg 300)

N = (the answer) The number of Management instances required to store the analytics data

$$N = \text{math.ceiling of } \frac{(C * E * 10^6 * 2 * 3)}{(1024^3 * D)}$$

Once your deployment is up and running if you find that you are beginning to run out of disk space for analytics data then there are two options available to you to add additional storage capacity;

1. Deploy one or more additional Management servers — the data will be distributed across the available servers which helps to spread the load for processing the data across a larger pool of infrastructure
2. Add additional disks to each of the existing Management servers to provide increase the amount of available storage on the existing servers.

As indicated by the table above, for very high API traffic rates you will generally need to apply a combination of both these options to provide the necessary storage capacity while constraining your deployment to a manageable number of servers.

Details on how to add an additional disk to a Management server (option 2) can be found in the Knowledge Center topic on "Adding a new data disk to a Management appliance"; **ibm.com**/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.install.doc/tapic_add_new_data_disk.html

5.5 Developer Portal

Session affinity	As described for UI traffic in section 5.3
Load balancing	As described for UI traffic in section 5.3
Persistence	Embedded variant of MySQL provides storage for the Drupal content-management items like blogs/forums, as well as local cache for API/Product and developer org/application data
Replication	Full copy of data in each instance, taking place over standard TCP/IP connectivity
Failover behaviour	Quorum/majority-based algorithm for determining whether an instance is permitted to process requests. All active instances can serve both read and write requests. Instances which cannot contact a quorum of expected peers will refuse to process any requests (reads or writes)
Deployment location	Private or protected zone. Not recommended for deployment in the DMZ. Additional security by isolating from other instances within the network zone.
Cluster sizing	Largely driven by HA and regional resilience – but also related to concurrent number of logged-in application developers

Table 8: **At a glance: Developer Portal**

The Developer Portal is a Drupal content management system that has been customized to meet the use cases required for managing APIs. Drupal sits on top of a MySQL-style database to provide its **persistence** so the resilience behaviours discussed here are generally inherited from the MySQL behaviour. Additionally, some aspects of the Portal configuration are persisted to the local disk of the Developer Portal nodes, and replicated around the cluster to provide resilience.

The primary purpose of the persistence in the Developer Portal service is to store the "content" that you configure within the Portal, or that application developers insert – for example customizations that you make to the theme of the site in order to reflect your corporate branding or desired look and feel (which is stored directly on the file system), or the content of blog posts, forum questions or other generated content that you may enable through the Drupal infrastructure (which is stored in the database).

Both the MySQL database and the file system storage are fully **replicated** across each instance, so that there a full copy in every server in the cluster. Both also use TCP/IP-based connections to carry out the replication activity, so only standard network connectivity is required – there is no dependency on multicast or similar advanced protocols.

As with the Management service, the Developer Portal instances are not security hardened to the extent that you would expect for it to be suitable to deploy the instances in the DMZ, so the correct **deployment location** for the Portal instances is in the protected or private zones. The Drupal content management system provides a wide spectrum of advanced capabilities that can be customized by the administrator, and so has an increased attack surface compared to a "single function" process, so some customers also choose to place the Portal instances into a restricted network zone that cannot communicate freely with other instances in the protected/private zone – this reduces the scope of the problem if a security exposure allows a malicious application developer user to get more powerful access than was intended. Note that you cannot isolate the Developer Portal instances entirely from the rest of the network as there is a requirement to talk back to the Management service but the set of instances/ports can be restricted to a permitted whitelist of endpoints.

Note:

Unlike the Management instances – which are sealed appliances – the Developer Portal instances do provide the ability for the user to reach the operating system on which the Portal infrastructure is running, for example via SSH. Having direct access to the operating system is very powerful from the perspective of being able to execute commands to configure and maintain the infrastructure but you are strongly recommended not to make any unsupported changes to the configuration since doing so will make it difficult or impossible for the IBM team to provide support should you need it, and such changes might be overwritten the next time an upgrade takes place. If you are in any doubt whether a change is permitted please contact the IBM support team before making the change.

5.5.1 Failover and Cluster Sizing

MySQL is different to some conventional databases in that every active server serves both read and write requests, so there isn't the concept of **failover** in the same way that there is for technologies where only a single "primary" instance can serve the write requests.

By default, however MySQL implements a quorum approach to determining whether a given instance is permitted to serve incoming requests – each instance will periodically poll its peers to determine whether they can be contacted, and only if the instance can see the majority of its peers will it permit itself to serve incoming workload. Any instances which cannot see the majority of its peers will refuse to serve any requests (read or write) and will return errors to the calling application – this is a defence mechanism which aims to avoid the equivalent split brain scenarios that we discussed in section 5.4.1.1 by ensuring that an instance will only process requests if it is connected to the majority of instances in the cluster.

This requirement for quorum can lead to some unusual deployment topologies as it encourages you to deploy different numbers of instances in each region – for example in a two-region deployment you might have three instances in one region and two in the other, effectively making the first region a primary region that can carry on transparently if the second region is lost. Like the Management node "main site" function discussed in section 5.4.1.1 the drawback of nominating a primary region is that an administrative command must be executed in the secondary region to allow it to become active if the primary region fails, because the two remaining instances will not be able to see a quorum of the expected servers. The specific administration commands to achieve this for the Developer Portal are described in the Knowledge Center topic on "Performing manual failover for Developer Portal servers";
ibm.com/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.install.doc/capic_install_portal_manual_failover.html

Controlling the failover behaviour of the system based on the number of nodes that are deployed into each region is not ideal and so there are also some configuration actions that you can apply to maintain failover behaviour while keeping an equal number of instances in each region as described in the Knowledge Center topic on "High Availability configurations for the Developer Portal". Note that even with an equal number of nodes in each location the commands described here are effectively nominating a "main site" and so in the event of the failure of the primary region it is still necessary to apply the administrative commands described in the previous paragraph.
ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.install.doc/capic_portal_ha_config.html

Cluster sizing for the Developer Portal service is determined by three main motivators;

- Number of application developers that will be concurrently logged in to the Portal
- Resilience and failover amongst the defined regions
- MySQL cluster size scalability

For many customers, the total number of application developers will be relatively small because they are internal to your enterprise or members of a small defined set of partners that develop applications on your behalf. It is generally only if you have large numbers of partners or are offering a truly public API programme where you expect to get thousands (or more) developers that you will need to consider adding extra instances to support the concurrent user load, over and above what you would already be considering to support your HA requirements.

The guidance given at the beginning of this whitepaper has a general recommendation to deploy multiple instances of a server type in each region so that the failure of a single instance does not cause a complete outage in that region, however in the case of the Portal there is also a constraint on the total number of instances in the cluster because the MySQL cluster replication does not scale well as the number of instances in the cluster increases beyond around five or six instances.

The following table gives some example topologies that embody the recommended trade-off between the total number of instances in the cluster, and the desire to give resilience to failure within a single region. Note that here we are talking about the number of regions in which the Developer Portal capability is deployed, which may not be all the regions covered by the API Connect deployment, as described in section 4.7.1.

Number of regions exposing the Developer Portal capability	Recommended instance configuration
One	Minimum 2 for high availability, preferably 3. Scale up to 5 depending on traffic requirements
Two	2 instances in each region, configuring the segment/weight commands as in the linked Knowledge Center topics above
Three	1 instance in each region, as described in the Knowledge Center topic above. Note that in this case the concern about the MySQL replication performance with large numbers of instances outweighs the desire to have the extra resilience of multiple instances within a region

Table 9: **Developer Portal instance layout based on number of deployment regions**

5.6 Business Services

Your business services are implemented outside the scope of API Connect, and may encompass a wide range of characteristics depending on the type of service and its implementation. The following table discusses some typical / desirable characteristics of your backend services as it relates to ideal integration with API Connect;

Session affinity	Ideally should not require session affinity, to maximize scalability and resilience
Load balancing	Round robin (no session affinity)
Persistence	Some form of persistence to store the business data, but style/type is specific to the service
Replication	Ideally the service instances are independent, but using a shared persistence store
Failover behaviour	No failover required (independent instances)
Deployment location	Private or protected zone. Not generally deployed in the DMZ.
Cluster sizing	Varies in line with backend traffic rate (which may be less than API traffic rate if response caching is in use)

Table 10: **At a glance: Business services**

5.7 Loopback Runtime

Session affinity	Loopback applications themselves are typically stateless as the data is persisted elsewhere, so no affinity requirements at the instance level
Load balancing	Round-robin distribution for both global and local load balancing. May configure affinity within the same region as the Gateway for performance efficiency
Persistence	No persistence inside the loopback application itself
Replication	Instances are independent of each other, no replication between them
Failover behaviour	No failover required (independent instances)
Deployment location	Private or protected zone. Not recommended for deployment in the DMZ.
Cluster sizing	Varies in line with API traffic rate

Table 11: **At a glance: Loopback runtime**

The loopback cluster in which the “Create/Run” parts of API Connect execute are generally stateless and have no **persistence** of their own, as they act as a bridge to an external data source within the enterprise. As such there is no **affinity** requirement and so round-robin **load balancing** is generally the best approach.

Since each instance is stateless there is also no need for **replication** of data between the instances, and so no real concept of **failover** beyond the behaviour of the load balancer to route around instances that may be unavailable at a given point in time.

These characteristics give significant flexibility in how the instances are "clustered" because from the perspective of serving runtime API traffic it doesn't matter whether the instances are part of a single cluster, multiple independent clusters, or un-clustered standalone instances. There is simplicity from an administration perspective in having a single cluster that is managed by a central coordinator such as a Collective so that it is only necessary to publish the loopback application once (and the coordinator distributes it to all the worker nodes), however for cross-region resilience you might have a separate cluster in each region - at the expense of having to publish the application twice.

As you would expect, **cluster sizing** in this case is driven by the volume of API calls to the APIs that are implemented by the loopback instances; starting with a minimum of two or three instances to provide high availability and then adding additional instances if the API throughput or concurrent request rate requires it.

5.8 Supporting Components

As well as the core components of API Connect that we have discussed above there are several supporting components which are critical to the successful and reliable operation of the deployment, and it is important to take these items fully into account when planning the resilience of your installation.

- **NTP —** There is a strong requirement that the various instances within the API Connect deployment should have a consistent view of the current time and retain that consistent view on an ongoing basis to enable the successful operation of the replication protocols that synchronize data between the different instances. Configuring NTP servers for each of the deployed instances is the most logical choice to achieve this requirement, but it is also important to make sure that the NTP servers themselves are giving a consistent view of the time, particularly if you are using different NTP servers in each deployment region
- **DNS —** DNS or hostname resolution is also likely to form an important part of your deployment infrastructure, whether it be for the public facing "branded" endpoints like api.mycompany.com and developer.mycompany.com, or for internal components of the system that provide routing to load balancers or the individual instances themselves. As highlighted in section 5.2 you may be using DNS as a simple mechanism to control routing of traffic to different regions so it is important that you can update those settings whenever you need to, and to have the change propagate reliably to your consumers in the expected timeframe
- **LDAP for User Interface access —** All the API Connect user interfaces (Cloud Manager, API Manager, Developer Portal) can be configured to authenticate against an external LDAP user registry if you wish them to. This has the advantage of letting users log in with their existing credentials to avoid having to remember separate passwords, but does mean that the LDAP server must be highly available and resilient. You should ensure you have considered the failure cases such as when the whole of a single region goes down – is the LDAP server replicated to the second region so that users can continue to log in successfully?
- **LDAP for API calls —** The same requirements described above apply when an LDAP server is used to provide authentication services for an API call – for example when the API Developer applies Basic Authentication or OAuth 2.0 authentication to an API. Since API calls are typically the highest importance aspect of the deployment the reliability and availability of the LDAP server is also critically important
- **SMTP —** There are various scenarios in API Connect in which an email notification can be generated and sent a user, for example when an application developer is registering themselves for the first time and needs to verify their identity by clicking the activation link, or when one of their applications is approaching the rate limit for its API calls, and the developer has configured that they wish to be notified before their application is prevented from making further API calls. As a result, it is important that the SMTP server you configure is also highly available and has a suitable failover approach so that it continues to be available from both/all deployment regions in the event of a failure

5.9 Example Deployment Topology

The following diagram shows an example deployment topology which highlights some of the aspects that we have discussed in this section, including the deployment location for each component. The example shows a single-region deployment however the network layout applies equivalently to any additional locations;

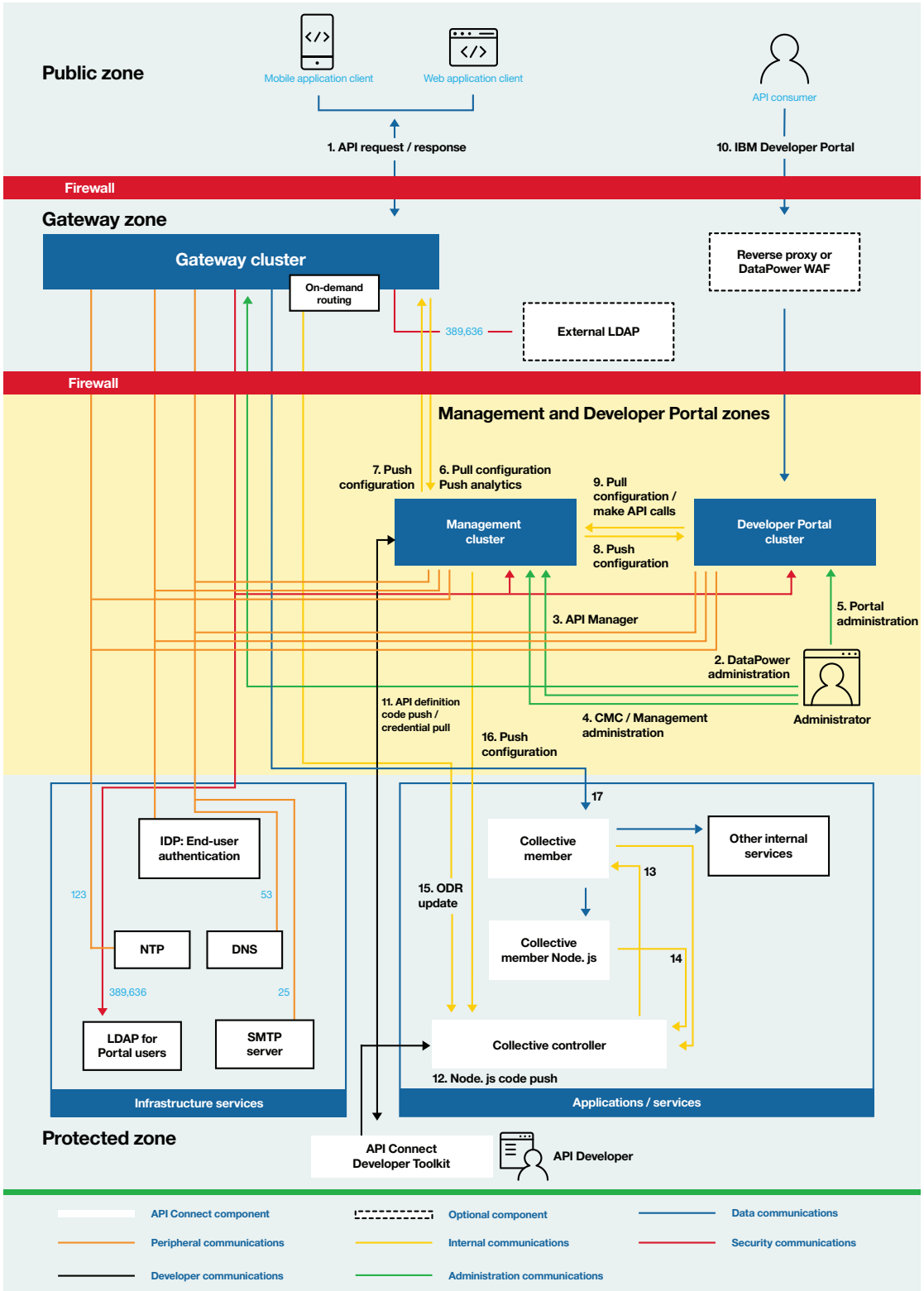


Figure 24: **Example deployment topology**

You can see the Gateway cluster deployed in an internet-facing network zone (ie the DMZ), but the Management and Developer Portal clusters behind an additional firewall. In this case, there is a separate network zone defined for the Management/Portal clusters which provides an additional level of protection in communication with the Protected zone where the backend services and applications reside.

The diagram also highlights some of the key data flows between the components of the system that we have discussed in the preceding sections, including some of the specific port numbers and protocols on which that communication takes place. More detail on the low-level networking requirements can be found in the accompanying KnowledgeCenter topic here, from which this diagram is taken;
ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.install.doc/overview_apimgmt_portreqs.html

6 Operational Topics

6.1 Disaster Recovery using Backup and Restore

The multi-region deployment approaches described in this whitepaper give the ability to deploy a highly available solution which can survive outages ranging from individual server failures to entire regions with little or no manual intervention.

Many customers consider the switchover to a second active or "hot standby" region to be a form of disaster recovery, where the disaster has taken out their first/primary data centre, however for the purposes of this whitepaper (as defined in section 2.2) we are defining disaster recovery as "the process of recovering the successful operation of the solution in the event of a total loss of the current infrastructure" – i.e. all of the existing deployed infrastructure has been lost, whether that be a single or multiple region deployment.

Under that definition, the process of disaster recovery is about standing up new infrastructure to support the service and restoring appropriate configuration/state from the original deployment into that new infrastructure, typically in the form of backups. You will need to identify how/where you will deploy the new API Connect infrastructure that will become your new deployment, and how the state from the original deployment will then be restored into that new infrastructure. Clearly this will be a time-consuming process so it is important to predict, test and document the elapsed time between making the decision to trigger the disaster recovery policy and the completion of the work required to begin serving your production workload once again.

As a reminder, there are three basic areas of persistent state in API Connect;

- Management service configuration database - APIs, Products, Apps, Subscriptions, Users etc
- Management service analytics data – API call event data and record of user actions
- Developer Portal – content management and filesystem data such as blogs, forums, themes

In preparing and testing your disaster recovery plan you will need to document what types of data you need to be able to recover in the event of a total loss of your existing infrastructure – the configuration database is effectively mandatory since that contains all the information about your APIs, subscribers etc, and it is likely that it is also important to recover the Developer Portal content since that includes any custom theme for your portal, and public collateral like blog posts, forum articles etc. You may however be willing to accept that historical analytics data need not be recovered in the event of a disaster, depending on the style of usage you make of it, and the frequency with which you expect to be triggering the disaster recovery policy.

The Gateway service contains largely transient state and so it may not be necessary to take backups of the Gateway servers themselves – when restoring the new deployment from your Management configuration database backup the necessary information about APIs and their subscribers will be pushed freshly to the Gateway instances you have deployed even if they were not part of the original deployment.

6.1.1 Management and Developer Portal

The Management service configuration database and Developer Portal database both have documented commands with which you can generate a backup file that can be later used to restore the configuration. Your operational infrastructure will need to control how frequently to take those backups, and arrange for them to be stored in an appropriate location so that the backups are not also lost when your disaster occurs – for example a different data centre than any of the deployed infrastructure components. The command to take the backup of the Management database includes the ability to push directly to an FTP/SFTP server, while in the Developer Portal case you will need to trigger a separate call to upload the generated backup to the fileserver.

Since backups are a point-in-time snapshot of the system state it is also important to consider the frequency with which you need to take backups – what duration of data are you willing to have lost if the disaster scenario occurs? This again will be a trade-off between the implications of losing data, the frequency of the occurrence, and the cost to implement the solution. An important case is that restoring to a point-in-time backup will roll back the creation of any new APIs, updates to existing APIs, or the registration of any new application developers/subscribers against your API estate. The latter can be undesirable particularly if they are external partners who thought they had registered but suddenly are no longer able to successfully invoke your APIs, but if the disaster scenario is only expected to be triggered once every 5-10 years (for example) then it may be acceptable to only take backups daily or weekly, with an acceptance of the risk and implications that brings.

Details on the backup/restore process for the Management service and Developer Portal are available in the Knowledge Center in the following topics;

- Management
  - **ibm.com**/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_backupcli_apimgmt.html
- Developer Portal
  - **ibm.com**/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.install.doc/tapim_portal_disaster_recovery.html

6.1.2 Analytics

By contrast to the discussion above on the Management service and Developer Portal, there is not currently a mechanism by which you can take an external backup of the data stored in the analytics repository, which includes historical data about API invocations, and records of user actions such as create/delete of APIs etc.

In v5.0.7.0 and later you can mitigate this issue by configuring your deployment to export analytics data to a 3rd party system such as an external ElasticSearch, Kafka or Syslog target. You might choose to do this in addition to having the data retained within the API Connect analytics store (which allows you to continue to use the analytics visualization capability inside API Connect) or instead of logging to the embedded analytics store. In the latter case your only store of the analytics data is in the external system and the requirements around resilience/failover are transferred to that system. More information about configuring external targets for analytics data can be found in the Knowledge Center here; **ibm.com**/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.cmc.doc/tapim_analytics_configuringanalytics.html

If the 3rd party export mechanism described above is not suitable for your scenario and it is critical that you are able to restore analytics data then the remaining option to explore is whether your virtualization platform provides backup/restore capabilities for the individual virtual machines that make up the deployment in a way that is suitable for implementing disaster recovery into a different data centre / hypervisor instance. Capabilities like taking a VMWare snapshot are generally intended only for short-term recovery points – for example taking a snapshot before an upgrade to be able to roll back in the event there is an issue, but your VMWare administrator may also be able to recommend an alternative technology approach that does work well for disaster recovery scenarios. Note that if you snapshot the entire virtual machine instance then it is not necessary to generate/restore the individual backup files that were described in the preceding section as that data will already be contained within the virtual machine image.

6.2 Custom Branding for APIs and Developer Portal Endpoints

For partner- or public-facing API programmes the API Gateway and Developer Portal endpoints form a part of the public face of your enterprise; applications developers come to your Developer Portal, typically over the Internet, to see what services you are offering and how they sign up to use them, and subsequently applications will embed the details of the Gateway endpoints that you expose. As a result, those endpoints are commonly "branded" using the same DNS name as your corporate internet presence, with a prefix to differentiate the two cases – common practice being to use the following;

- https://developer.mycompany.com (for Developer Portal)
- https://api.mycompany.com (for API calls)

Successfully implementing this style of custom branded endpoint requires three main types of action which are carried out by different user roles within your organization;

1. **Configure the API Connect endpoint to use your custom hostname** (eg API endpoint or Developer Portal URL in the Catalog settings respectively) - this is the responsibility of the catalog administrator inside the provider organization (eg someone who has the role of Owner, Administrator, Publisher, or custom role that has been assigned the "edit catalog" permission)
2. **Configure your deployment to present an appropriate TLS certificate** matching the custom hostname when a request reaches your infrastructure so that the caller knows they can trust the endpoint – depending on where you have chosen to terminate incoming TLS connections this might be the responsibility of your load balancer administration team, or Cloud Manager administrator
3. **Configure your DNS setup** so that your custom hostname resolves to the correct IP address(es) when the caller does their DNS lookup, which is handled by your DNS domain owner or networking team

The following diagram illustrates these three steps;

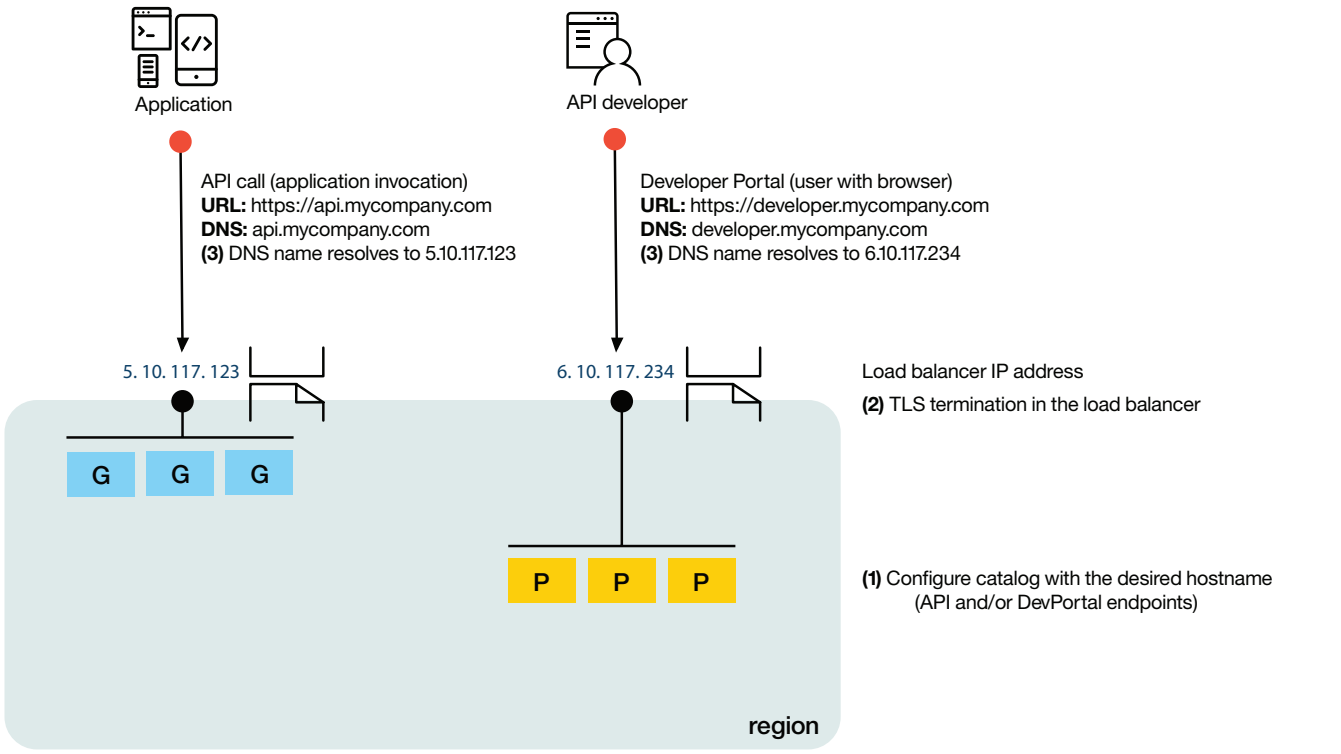


Figure 25: **Steps involved in custom branding**

Note that for on-premises deployments (which are managed by the customer) the three tasks above will be handled within your own team, but for deployments in Bluemix Public and Bluemix Dedicated steps 2 and 3 will be the joint responsibility of the customer organization and the IBM API Connect / Bluemix Operations team;

- The TLS certificate will be generated by the customer organization (as the owner of the "mycompany.com" domain) typically through a trusted certificate authority such as DigiCert so that the endpoint will automatically be trusted by browsers and common application client environments. That TLS cert will then be given to the IBM API Connect Operations team who will deploy it to the Bluemix infrastructure so that it is presented when an application connects to that endpoint
- The IBM API Connect Operations team will share a DNS name such as "branding.api.eu.apiconnect.ibmcloud.com" which is configured to resolve to the necessary IP address, and the customer's DNS administrator configures a CNAME from the custom hostname "api.mycompany.com" to the IBM-provided DNS name

6.2.1 API Call Branding

The steps required for configuring the Catalog with a custom hostname for API calls are described in the following Knowledge Center topic – specifically you will be setting the "Custom Gateway URL" property to your branded hostname;
ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.apionprem.doc/create_env.html

In v5.0.4.0 and later the Gateway uses the incoming hostname from your API call to detect which Provider Organization and Catalog the request is intended for, so the full path of your API call will be as follows;

- Unbranded:
  - https://api.internalname.com/<providerOrg>/<catalogName>/<apiRoot>/<operation>
- Branded:
  - https://api.mycompany.com/<apiRoot>/<operation>

Note that prior to v5.0.4.0 this automatic catalog detection was not in place so it was necessary to configure a separate component in front of the Gateway (for example using advanced features of the load balancer) to detect the incoming hostname and rewrite the URL so that it was in the "unbranded" form when the request was forwarded on to the Gateway.

It is common to carry out the TLS termination in the load balancer as this is a corporate networking requirement in many customer organizations, but also API Connect does not provide the ability to configure different TLS certificates per Catalog so an external component like a load balancer must be used in front of the Gateway to present the TLS certificate if there is going to be more than one Catalog in the deployment.

6.2.2 Developer Portal Branding

The hostname for the Developer Portal is configured as the "Portal URL" when the Developer Portal is enabled within the Catalog settings as described in the following Knowledge Center topic. Note that Portal URL is not the same as the "Hostname for Developer Portal API Calls", the latter also being configured in the Catalog settings but having a different purpose.
ibm.com/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.apionprem.doc/create_env.html

It is best to carefully consider your choice of Portal URL before you create the Catalog as changing it after the portal has been created requires several steps to update the Drupal configuration as well as the Management service view.

As with the API Gateway it is common to carry out TLS termination in the load balancer or other component in front of the Developer Portal cluster as API Connect does not provide the ability to configure an individual TLS certificate per developer portal site.

It is important to note that the Portal URL used by the customer to access the portal site must exactly match what is configured in API Connect, and thus the Drupal infrastructure – the hostname/URL cannot be changed as the request passes through the network between the user and the portal cluster since the Drupal infrastructure uses its internal view of the URL to generate things like activation links, password reset notification emails and authentication endpoints for use in OAuth, OpenID Connect etc.

7 Summary

Congratulations - you've made it to the end of this whitepaper on API Connect deployment!

Over the course of the preceding sections we have discussed a series of topics which will hopefully have given you a detailed insight into the way that API Connect works and how it can be deployed into a variety of architectures to support production level reliability and availability;

- In section 2 we introduced API Connect, the major components that comprise the solution, and introduced the necessary terminology and logical concepts to give you the context for the rest of the whitepaper
- In section 3 we talked about the key user scenarios, related failure scenarios and the questions you should ask yourself as part of the process of determining your availability and resilience requirements
- Section 4 introduced typical deployment patterns starting with a minimal development installation and progressing through high availability configurations within a single region up to multi-region global deployments and various cloud/hybrid scenarios
- In section 5 we looked in detail at each of the major deployment components of the system and the aspects of their operation that affect the way that you deploy or manage the solution at runtime
- Lastly in section 6 we covered some related operational topics including disaster recovery and custom branding

We hope that you have found this whitepaper a useful reference and that it helps you to successfully deploy API Connect in large scale production scenarios for your enterprise. As always, we welcome your comments and feedback via our DeveloperWorks Community here – tag your questions in the Community space with "API Connect";
developer.ibm.com/apiconnect

7.1 About the Author

Matt Roberts is an Architect within the IBM API Connect product development team where he focuses on production quality of service scenarios and the deployment and management of API Connect in Bluemix/SaaS.

Matt is based in the IBM Hursley Lab in the United Kingdom and prior to API Connect held a range of architect and technical leadership roles across the IBM integration portfolio including API Management, Cast Iron, WebSphere Application Server/ESB and IBM MQ. Matt also spent three years as a Senior IT Consultant providing leading edge on-site IT consultancy to IBM customers across the UK and Europe as part of the IBM Software Services for WebSphere team.

You can find Matt on LinkedIn here:
www.linkedin.com/in/matrober

Matt would like to thank the members of the broader API Connect team who provided input to this whitepaper and reviewed the drafts before publication including Quentin Presley, Evan Jardine-Skinner, Shiu-fun Poon, Layne Miller, Chris Dudley, Mark Nesbitt, Dan Whitacre and Jim Thorpe. Credit also goes to Lara Baker-Olin for the example deployment topology diagram shown in section 5.9.

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
June 2017

Please Recycle

IBM **API** Connect

IGW12361-USEN-00