

# Autonomous Cloud Lab

Participant Guide

---

## Username and Passwords

The following usernames and passwords will be used throughout the lab. You will be assigned a username. Passwords are all the same.

Bastion Host IP	Username	Password
Common IP Address		Common Password

## Table of Contents

Username and Passwords .....	2
Table of Contents .....	3
Autonomous Cloud Management.....	5
Agenda and Class Information.....	6
Housekeeping .....	6
Ground Rules .....	6
Cloud-Native Concepts.....	7
What is a Microservice?.....	7
What is a Container Image?.....	7
Comparing Containers and Virtual Machines.....	7
What is a Container Registry?.....	7
What is Kubernetes?.....	9
Kubernetes Architecture.....	9
What is OpenShift? .....	9
Monolith to Microservices.....	11
Limitation of monolithic applications .....	11
Limitations of Monoliths have given rise to Microservices.....	11
Building Environment Zero .....	20
Sockshop Application .....	20
Architecture .....	20
Lab Setup .....	20
Developing Microservices.....	28
Steps to Deploy a Microservice on Kubernetes .....	28
The Liveness Probe and Readiness Probe.....	29
When should you use liveness or readiness probes? .....	29
Monitoring as a Service .....	34
Full-Stack: Monitoring as a Platform.....	34
Process Groups.....	36
Process Group Best Practices.....	36
Performance as a Service .....	44
Use Cases – Key Takeaways .....	44
Load Testing Integration:.....	44

Ensure Proper Tagging and Deployment Events .....	45
<b>Production Deployments.....</b>	<b>54</b>
Deployment Strategies .....	54
Istio architecture .....	56
Istio components.....	56
<b>Runbook Automation and Self-Healing.....</b>	<b>65</b>
<b>Unbreakable Delivery Pipeline.....</b>	<b>71</b>
Hands-on Building the Unbreakable Delivery Pipeline .....	71
<b>Virtual Operations .....</b>	<b>80</b>
Using the Dynatrace API .....	80
Using Dynatrace Davis.....	80
<b>Autonomous Cloud Lab Questions .....</b>	<b>86</b>
<b>ACL Key Takeaways .....</b>	<b>86</b>
<b>Contact Information .....</b>	<b>86</b>

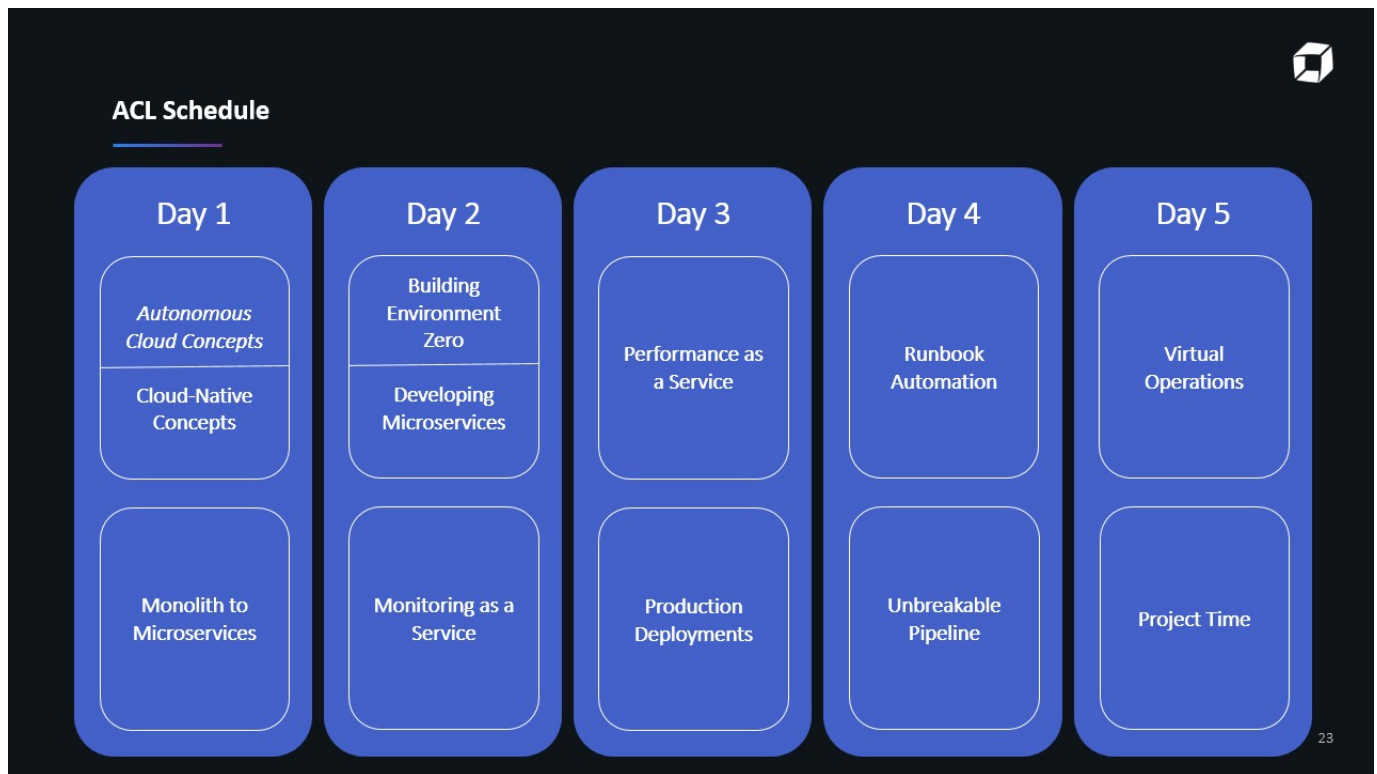
## Autonomous Cloud Management



As the instructor introduces Autonomous Cloud Management, think about the following questions and record your thoughts.

What challenges do you face within your organization?
How many apps can you manage?
What are your goals?

## Agenda and Class Information



### Housekeeping

- Facilities instructions
- Wi-Fi passwords
- Parking lot

### Ground Rules

- **Eliminate Distractions:** Silence cell phones, email notifications, messaging and social media.
- **Express any challenges:** If you have an immediate question or get held up completing any activity, let us know.
- **Exercise patience:** Some of the tools used in this lab may be new for some but not for others. We will move along quickly, but please be patient so that we can all be successful.

## Cloud-Native Concepts

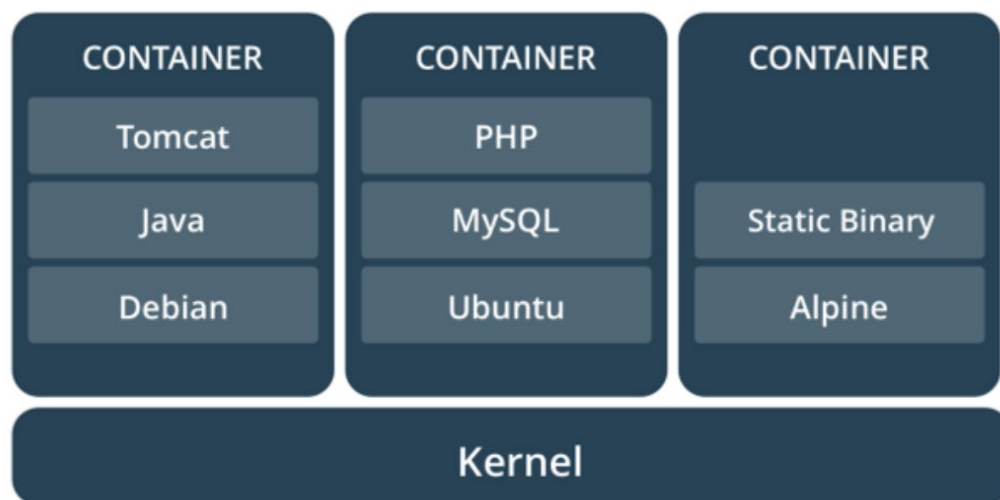
In this module you will learn a definition of Cloud-native concepts such as *Microservice*, *Container* (and *Container Image*), *Container Registry*, and *Kubernetes*.

### What is a Microservice?

An architectural style that structures an application as a collection of loosely coupled services. Services are fine-grained, and their protocols are lightweight.

How could microservices help with your development?

### What is a Container Image?



A container image is a lightweight, stand-alone, executable package of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings.

### Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.

### What is a Container Registry?

In a Container Registry you can store, manage, and secure your Docker container images.



## Lab activity:

### Working with Container Images and Containers



The goal of this lab activity is to provide you with hands-on experience with container registries. To reach this goal, you will:

- Create a Dockerfile
- Build and tag a container image
- Run a container
- Pull a container image from a container registry

#### Instructions

If you have not, clone or download the lab GitHub repository to your local machine.

This lab is found in the GitHub workshop repository:

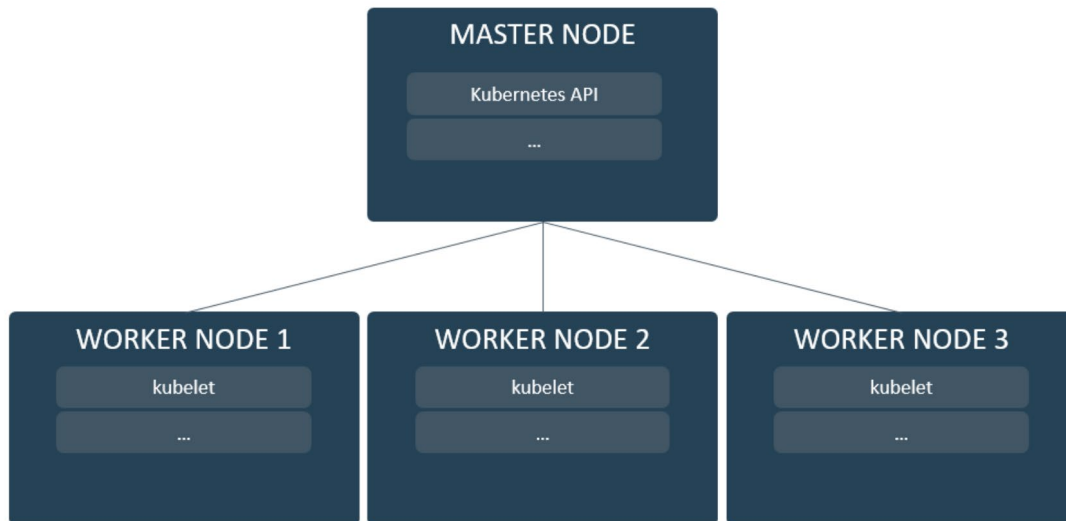
[course-repository/02\\_Cloud\\_native\\_Concepts/01\\_Working\\_with\\_Containers/](#)

#### Notes:

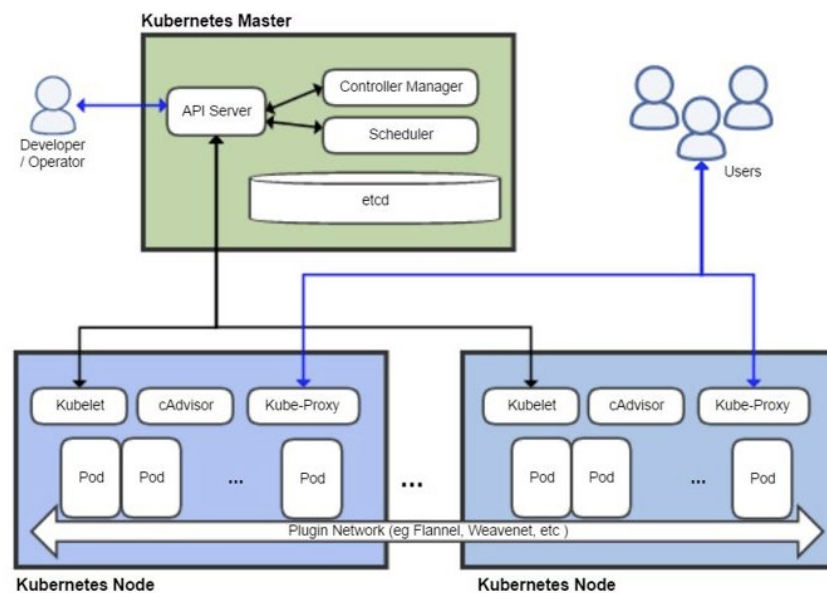


## What is Kubernetes?

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.



## Kubernetes Architecture



## What is OpenShift?

Red Hat OpenShift Origin is an open source container application platform based on the Kubernetes container orchestrator for enterprise application development and deployment.

## Summary Statements

Can you complete the following summary statements about cloud-native concepts?

- A \_\_\_\_\_ structures an application as a collection of loosely coupled services that enable continuous delivery and deployment.
- A \_\_\_\_\_ is a lightweight, stand-alone, executable package of software that includes everything needed to run it. When the image is run it becomes a \_\_\_\_\_.
- A \_\_\_\_\_ allows you to store, manage and secure docker container images.
- \_\_\_\_\_ is an open-source system used to automate deployment, scaling, and the management of container applications.
- \_\_\_\_\_ is an open source container application platform based on the Kubernetes container orchestrator for enterprise application development and deployment.

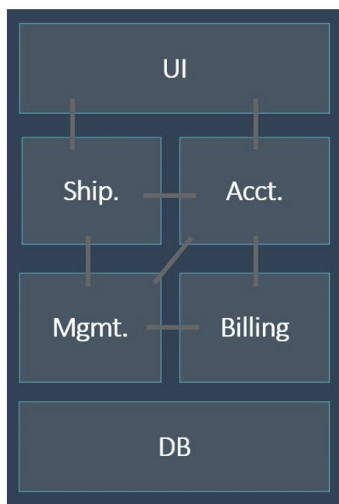
## Questions

Do you have any questions about how these cloud-native concepts could be applied within your organization? Jot them down here, there will be an opportunity to discuss with a Dynatracers.

## Monolith to Microservices

In this module you'll learn how to migrate a monolithic application to the cloud and how to fearlessly break it up into microservices. Therefore, we want to walk you through the different stages of identifying and extracting a microservice, as well as strangling it around its origin – the monolith. For this purpose, the module provides step-by-step instructions and labs showing the best practices we have identified for migrating a monolith to the cloud.

### Limitation of monolithic applications



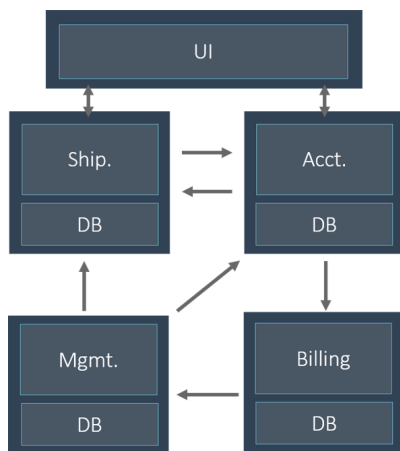
**Agility** – Rebuilding the whole application takes a decent amount of time

**Scalability** – Scaling a monolith happens in both directions: vertically as well as horizontally - causing unused resources

**DevOps Cycle** – Continuous delivery (high frequency of deployments) fails due to high build time

Availability, fault tolerance, and resiliency

### Limitations of Monoliths have given rise to Microservices



**Agility** - Scope changes can be done in one microservice - other microservices are not impacted from these changes

**Scalability** - Individual components can scale as needed

**DevOps Cycle** - Since each component operates independently, continuous delivery cycle reduces



## Lab activity:

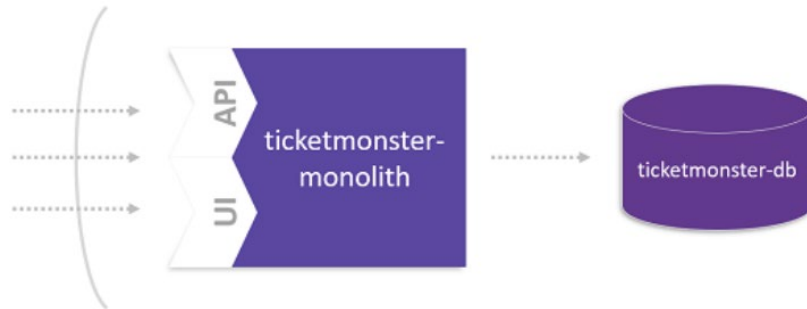
### Monolith to Microservices

#### Part 1 of 7

### Lift-and-Shift TicketMonster

The goal of this lab activity is to provide you with hands-on experience with moving a monolithic application to a cloud platform applying lift and shift concepts. To reach this goal, you will:

- Create a MySQL service for monolith
- Push application to OpenShift
- Bind MySQL service to monolith



### Instructions

Make sure you have:

1. [OpenShift CLI](#)
2. The OpenShift Cluster, username, and password for use with this lab (ask instructor)
3. The Dynatrace tenant, with username, and password (ask instructor)

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/1\\_Lift-and-Shift\\_TicketMonster/](#)

### Notes:



## Lab activity:

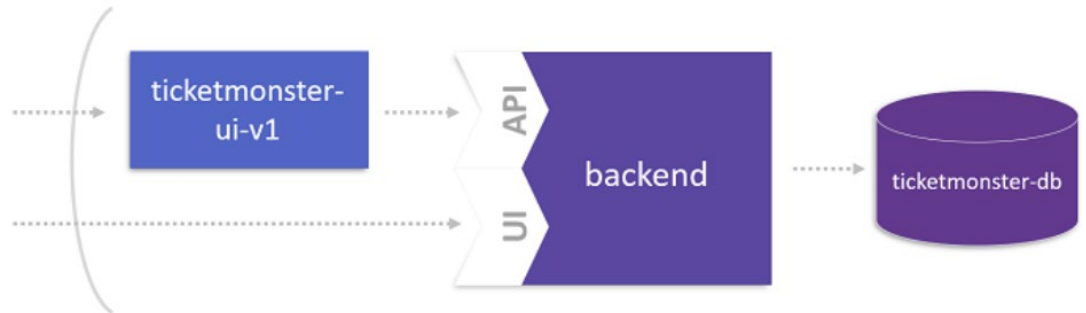
### Monolith to Microservices

#### Part 2 of 7

### Extract the User Interface from the Monolith

The goal of this lab activity is to launch the first microservice used to separate the user interface from the monolithic application. To reach this goal, you will:

- Define a new route to the monolith
- Decouple the UI from the monolith
- Test the UI that hits the monolith



#### Instructions

This lab is found in the GitHub workshop repository:

*course-*  
*repository/03\_Monolith\_to\_Microservices/2\_Extract\_UI\_From\_Monolith*

#### Notes:



## Lab activity:

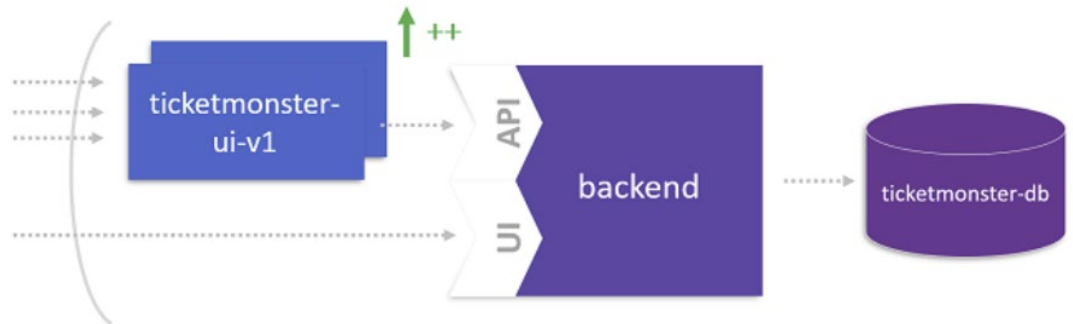
### Monolith to Microservices

#### Part 3 of 7

### Generate Load on the User Interface

The goal of this lab activity is to run load generation scripts that simulate navigation scenarios. To reach this goal, you will:

- Build docker image
- Run container and start script



#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/3\\_Generate\\_Load\\_on\\_UI](#)

#### Notes:



## Lab activity:

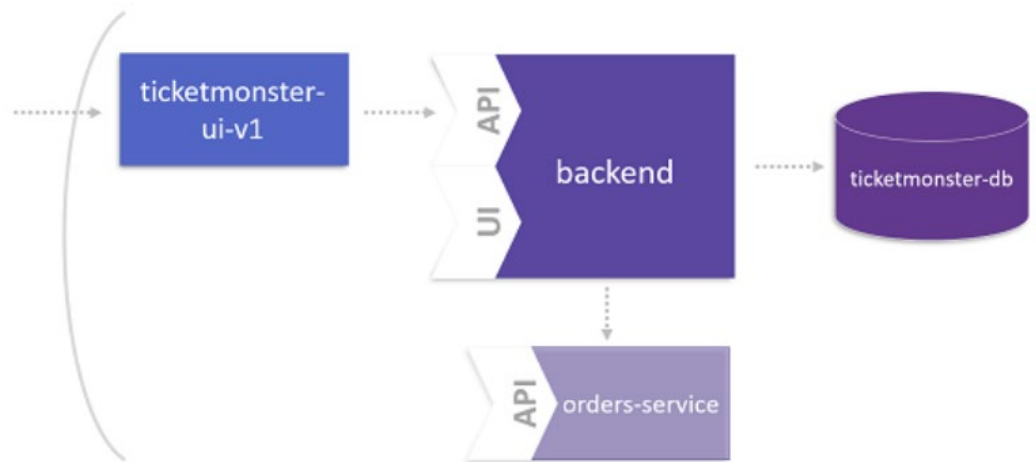
### Monolith to Microservices

#### Part 4 of 7

#### Identify a Microservice

The goal of this lab activity is to virtually break a monolithic application using Dynatrace Service Detection to define an entry point. To reach this goal, you will:

- Define custom service entry points
- Restart pods to activate custom service detection
- Book a ticket on TicketMonster
- Consider service flow in Dynatrace



#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/4\\_Identify\\_a\\_Microservice](https://github.com/dynatrace-workshop/course-repository/tree/main/03_Monolith_to_Microservices/4_Identify_a_Microservice)

#### Notes:



## Lab activity:

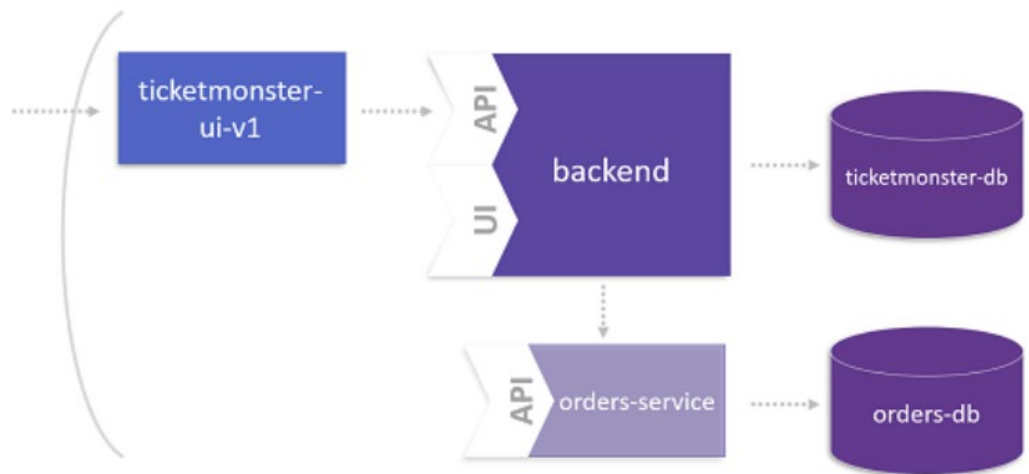
### Monolith to Microservices

#### Part 5 of 7

### Identifying the Domain Model of the Microservice

The goal of this lab activity is focused on the data management of the microservice created. The microservice should consume legacy data from the monolith while persisting new data in its own database. To reach this goal, you will:

- Use Dynatrace to learn more about the Domain Model
- Create a database for the microservice
- Setup the database



### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/5\\_Domain\\_Model\\_of\\_Microservice](#)

### Notes:





## Lab activity:

### Monolith to Microservices

#### Part 6 of 7

### Deploy the Microservice

You have identified the microservice OrderService that has its own code repository and defines its own domain model. The goal of this lab activity is to direct the backend service to intercept all incoming requests and forward synthetic or live traffic to OrderService. To reach this goal, you will:

- Deploy the Microservice
- Deploy a new backend version (v2) of the monolith
- Switch feature Flag and test your microservice

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/6\\_Deploy\\_the\\_Microservice](#)

#### Notes:



## Lab activity:

### Monolith to Microservices

#### Part 7 of 7

#### Clean up



The goal of this lab activity is to clean up the OpenShift project and Dynatrace. To reach this goal, you will::

- Delete services and pods on OpenShift
- Delete management zone and Custom Service Detection in Dynatrace

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/03\\_Monolith\\_to\\_Microservices/9\\_Clean\\_up](#)

### Notes:

## Summary

Dynatrace can be leveraged in this Monolith to Microservices Journey to:

- Get Dependency Information
- Detect Service Endpoints, Usage & Behavior
- Understand Service Flow per Endpoint
- Finding Entry Points with CPU Sampling Data
- Define Custom Service Entry Points

## Additional Learning

Dynatrace Blog post: [Fearless Monolith to Microservices Migration – A guided journey](#)

## Questions

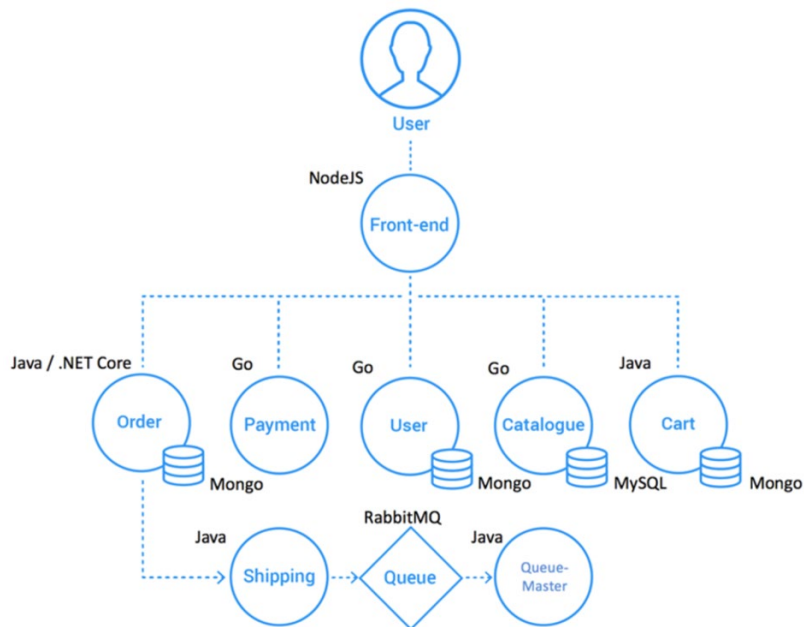
Do you have any questions about moving your monolith application to microservices? Jot them down here and share them with your instructor.

## Building Environment Zero

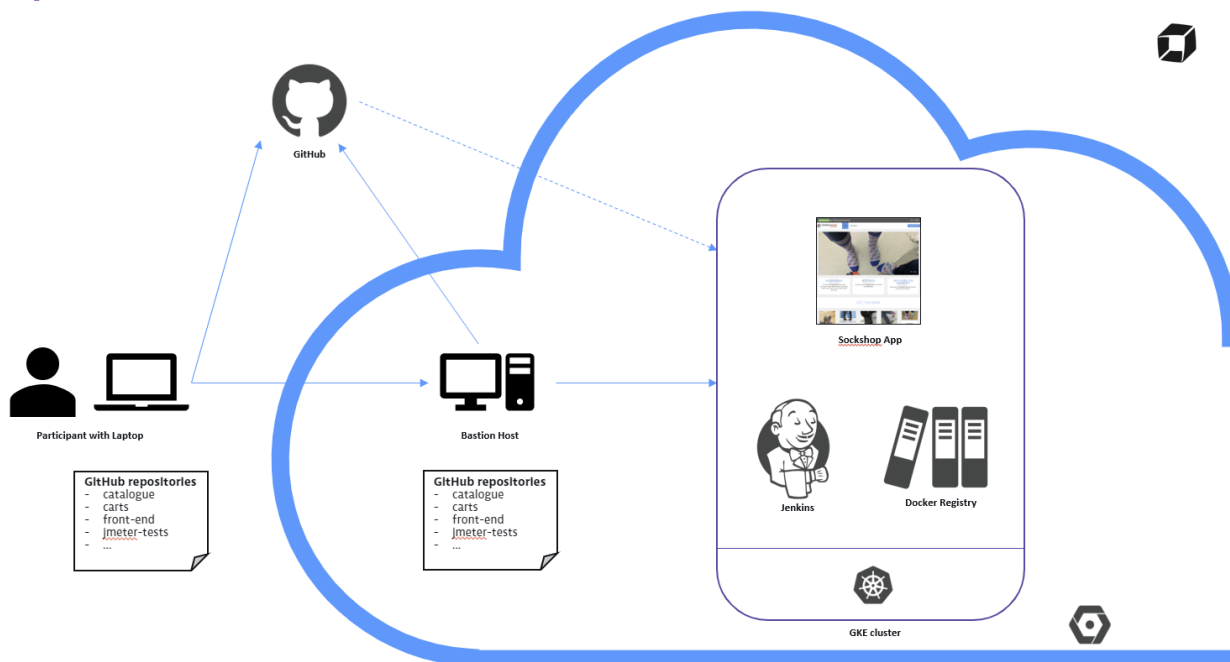
### Sockshop Application

<http://socks.weave.works>

### Architecture



### Lab Setup





## Lab activity:

### Building Environment Zero

#### Part 1 of 6

#### Check Prerequisites



The goal of this lab activity is to provide you with hands-on experience setting up your own GKE cluster to use for the duration of the lab. To reach this goal, you will:

- Setup Jenkins and a Docker Registry
- Fork several GitHub repositories
- Configure build pipelines for all microservices in the SockShop application hosted in the GitHub repositories

#### Instructions

To get started, checked that you have access to the Bastion Host and that kubectl is configured. Follow the steps found here:

[course-repository/04\\_Building\\_Environment\\_zero/1\\_Check\\_Prerequisites](#)

#### Notes:



## Lab activity:

### Building Environment Zero

#### Part 2 of 6

### Fork GitHub Repositories

This lab activity will get you started by creating your own GitHub repository. To do this, you will:

- Need your GitHub user account and password
- Login to [GitHub](#)
- Create a GitHub organization
- Clone a repository on the Bastion host

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/*

[04\\_Building\\_Environment\\_zero/2\\_Fork\\_GitHub\\_Repositories](#)

#### Notes:



## Lab activity:

### Building Environment Zero

#### Part 3 of 6

### Deploy Docker Registry



The goal of this lab activity is to use the Docker registry to save the build artifacts, i.e. Docker containers. The registry allows us to host multiple versions of an artifact. To reach this goal you will:

- Create Kubernetes namespaces
- Create a PersistentVolumeClaim (PVC) where Docker images of the registry will be stored
- Create the service and the deployment for the Docker registry

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/](#)

[04\\_Building\\_Environment\\_zero/3\\_Deploy\\_Docker\\_Registry](#)

### Notes:



## Lab activity:

### Building Environment Zero

#### Part 4 of 6

#### Deploy Jenkins



For this lab, we will use Jenkins CI/CD pipeline tool. The goal of this lab activity is to deploy Jenkins as a Kubernetes service. To reach this goal, you will:

- Create a PersistentVolumeClaim (PVC) for Jenkins
- Create and deploy the Jenkins service
- Login to and configure Jenkins

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/04\\_Building\\_Environment\\_zero/4\\_Deploy\\_Jenkins](#)

#### Notes:





## Lab activity:

### Building Environment Zero

#### Part 5 of 6

#### Trigger Build Pipelines



The goal of this lab activity is to prepare the dev, staging, and production namespaces in Kubernetes, so that the services find the infrastructure components that they need. Then, the build pipelines can be triggered. To reach this goal, you will:

- Setup databases and Rabbit MQ
- Trigger build pipelines in Jenkins

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [04\\_Building\\_Environment\\_zero/5\\_Trigger\\_Build\\_Pipelines](#)

#### Notes:



## Lab activity:

### Building Environment Zero

#### Part 6 of 6

### Clone GitHub Repositories

As the last activity in this lab, you will clone the previously forked GitHub repositories (Step 2) to your local environment, for editing in Visual Studio Code.

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/ 04\\_Building\\_Environment\\_zero/ 6\\_Clone\\_GitHub\\_Repositories](#)

#### Additional Learning

[kubect! Cheat Sheet](#)

#### Notes:

#### Questions

Do you have any questions about GitHub, Jenkins, or any of the steps performed during these lab activities? Jot them down here and share them with your instructor.

--

## Questions

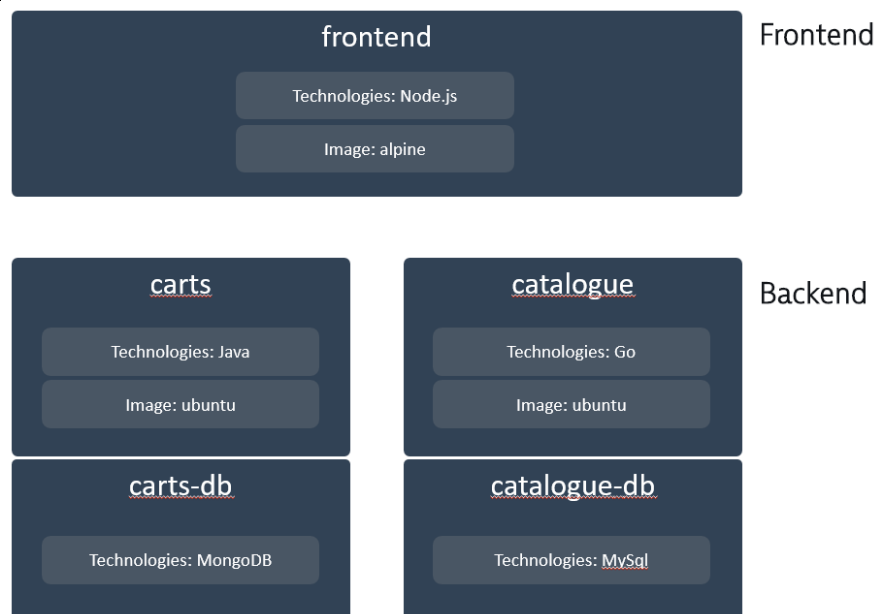
Do you have any questions about the steps taken to “Build Environment Zero”? Jot them down here and share them with your instructor.

## Developing Microservices

In this module you will learn more about the structure of a microservice from a code perspective.

Recalling the information from Day 1, how would you describe the microservices architecture?

How would you define the endpoints?



## Steps to Deploy a Microservice on Kubernetes

- Step 1: Create a configuration (.yaml Files)
- Step 2: Create a deployment configuration (carts-dep.yaml)
- Step 3: Create a service configuration (carts-svc.yaml)
- Step 4: Execute the configuration (.yaml Files)

## The Liveness Probe and Readiness Probe

Generally, a Probe is a diagnostic performed periodically by the kubelet on a Container. The kubelet can optionally perform and react to two kinds of probes on running containers:

**livenessProbe:** Indicates whether the Container is running

**readinessProbe:** Indicates whether the Container is ready to service requests

## When should you use liveness or readiness probes?

- If the process in your Container can crash on its own
- If you'd like your Container to be killed and restarted if a probe fails
- If you'd like to start sending traffic to a Pod only when a probe succeeds, specify a readiness probe.
- If your Container needs to work on loading large data, configuration files, or migrations during startup, specify a readiness probe.



## Lab activity:

### Developing Microservices

#### Part 1 of 3

### Deep Dive into Carts Service

The goal of this lab is to provide you with hands-on experience structuring a microservice from a code perspective. To reach this goal, you will:

- Deep dive into the carts service from a code perspective
- Commit a change of carts and re-deploy it into the environment
- Create a release branch to trigger the pipeline for the staging environment

In part 1, you'll first investigate the code structure of a microservice and then deploy it to a Kubernetes cluster to see the service in action. To do this you will:

- Become familiar with the carts microservice from a code perspective
- Deploy the service to a Kubernetes cluster

### Instructions

This lab is found in the GitHub workshop repository:

*course-*  
*repository/05\_Developing\_Microservices/01\_Deep\_Dive\_into\_Carts\_Service*

### Notes:



## Lab activity:

### Developing Microservices

#### Part 2 of 3

### Deploy the Carts Service to Dev



The goal of this lab activity is to learn how the Jenkins pipeline is designed to build, deploy, and test a microservice after pushing a source code change to its repository. To do this you will:

- Modify the carts service
- Build a new version in Jenkins

Stage View

Average stage times: (Average full run time: ~14min 27s)		Declarative: Checkout SCM	Maven build	Docker build	Docker push to registry	Deploy to dev namespace	Run health check in dev	Run functional check in dev	Mark artifact for staging namespace	Deploy to staging
Oct 31 09:32 commit		1s	22s	6s	7s	8s	3min 26s	29s	55ms	53ms
		486ms	17s	7s	8s	10s	2min 23s	23s		

### Instructions

This lab is found in the GitHub workshop repository:

*course-*  
*repository/05\_Developing\_Microservices/02\_Deploy\_Microservice\_to\_Dev*

### Notes:



## Lab activity:

### Developing Microservices

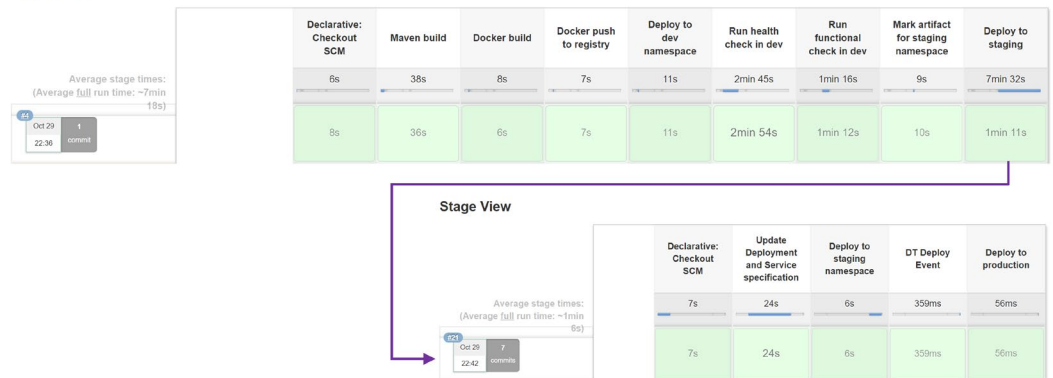
#### Part 3 of 3

### Deploy the Microservice to Staging

The goal of this lab activity is to learn how to use the Jenkins pipeline to release a microservice to the staging environment. To do this you will:

- Create a new release
- Build a new release in Jenkins

Stage View



### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/05\\_Developing\\_Microservices/03\\_Deploy\\_Microservice\\_to\\_Staging](#)

### Notes:



## Summary

- A Microservice Architecture structures an application as a collection of loosely coupled services that help enable continuous delivery and deployment.
- kubelet probes:
  - livenessProbe: Indicates whether the Container is running.
  - readinessProbe: Indicates whether the Container is ready to service requests.
- In the labs we:
  - Explored the code structure of the carts service and deployed it to a Kubernetes cluster.
  - Committed a change and re-deployed it to the development environment using the Jenkins pipeline.
  - Used the Jenkins pipeline to release the microservice to the staging environment.

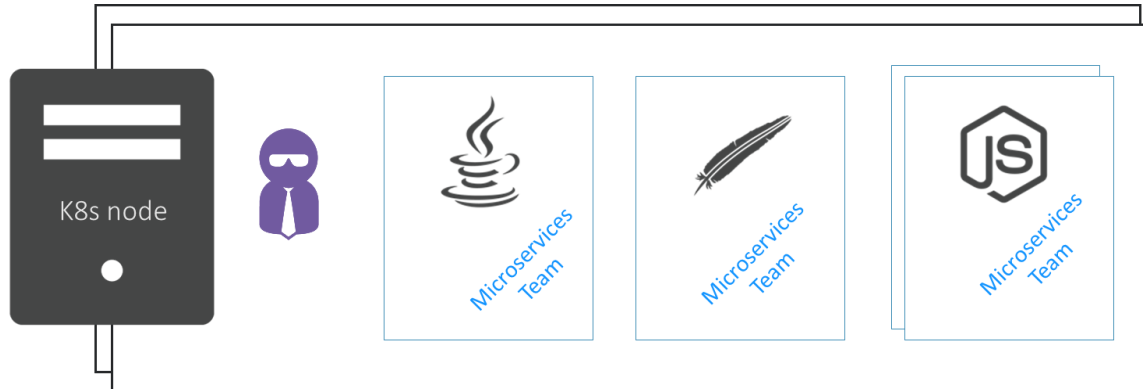
## Questions

Do you have any questions about developing microservices, Kubelet probes, or about any of the steps performed during these lab activities? Jot them down here and share them with your instructor.

## Monitoring as a Service

In this module, you will learn how to move from a long manual process of configuring monitoring for a new app to fully automated monitoring.

### Full-Stack: Monitoring as a Platform



- Deploy one agent per host (kubectl create)
- Auto instrumentation of containerized microservices
- Auto distributed transaction tracing
- AI based root cause analytics

What is an Operator in Kubernetes?
What is Dynatrace One Agent Operator?



## Lab activity:

### Monitoring as a Service Step 1 of 7

### Instrument Kubernetes Cluster with Dynatrace OneAgent Operator



In this lab you'll instrument the Kubernetes Cluster (from the Building Environment Zero lab) with Dynatrace OneAgent, which automatically monitors all your processes, containers, services, applications, and end-users. To do this you will:

- Define the Cluster Role Binding
- Get the Dynatrace API URL, API Token, and PaaS Token
- Rollout Dynatrace OneAgent Operator
- Enable Go monitoring

Then, you will explore the automated monitoring result in Dynatrace.

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [06\\_Monitoring\\_as\\_a\\_Service/01\\_Instrument\\_Cluster\\_with\\_Dynatrace\\_OneAgent](#)

#### Additional Learning

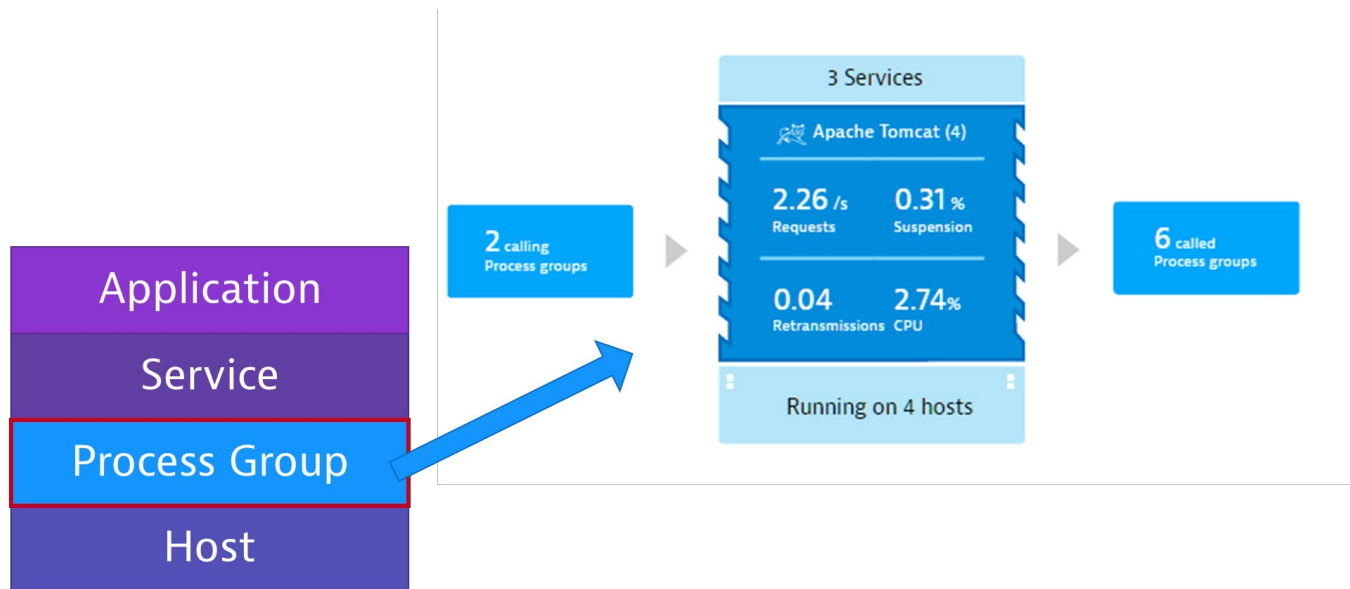
[What is OneAgent Operator?](#)

#### Notes:

## Process Groups

Process groups are clusters of processes that belong together. They run on the same software between hosts and services.

- Used as configuration points
- Creates continuity within process and plugin metrics
- Automatic for all process types



What are characteristics of process groups?
How can process groups be customized?

## Process Group Best Practices

- Unique enough to easily identify the process role
- No version numbers
- "Environments" split out
- Don't manually change the name unless 100% certain it will be consistent through deployments
- Configure Process Groups BEFORE Services if possible
- ALL rule changes require a process restart to take effect



## Lab activity:

### Monitoring as a Service Step 2 of 7

### Pass and Extract Meta-Data for each deployed Process or Container



In this lab you learn which meta-data is captured automatically, how to pass custom meta-data and how you can use this meta-data to influence process group detection and automated tagging. To do this you will:

- Add the DT\_TAGS and DT\_CUSTOM\_PROP environment variables
- Redeploy the carts service

Then, you will be able to see the tag and custom meta-data in Dynatrace.

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/06\\_Monitoring\\_as\\_a\\_Service/02\\_Pass\\_Extract\\_Meta-Data\\_for\\_Process\\_or\\_Container](#)

#### Additional Learning

[Define your own process group metadata](#)

[Adapt the composition of default process groups](#)

[Automated rule-based tagging for services](#)

#### Notes:



## Lab activity:

### Tagging of Services and Naming of Process Groups Step 3 of 7

Pass and Extract  
Meta-Data for each  
deployed Process  
or Container

In this lab you'll learn how to automatically apply tags on the service level. This allows you to query service-level metrics automatically based on meta-data that you have passed during a deployment. To do this you will:

- Create a Naming Rule for Process Groups
- Create Service Tag Rules

Then, you will be able to see the carts service tagged in Dynatrace.

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [06\\_Monitoring\\_as\\_a\\_Service/](#)  
[03\\_Tagging\\_and\\_Naming\\_of\\_Services](#)

#### Notes:



## Lab activity:

### Monitoring as a Service Step 4 of 7

### Push Deployment & Configuration Change Events to Dynatrace



In this lab you'll learn how to push deployment and configuration events to Dynatrace using the Performance Signature Plugin in Jenkins. To do this you will:

- Configure Jenkins
- Extended the carts and staging pipelines

Then, you will be able to see the deployment events in Dynatrace.

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [06\\_Monitoring\\_as\\_a\\_Service/04\\_Push\\_Events\\_to\\_Dynatrace](#)

#### Additional Learning

[Dynatrace Events API](#)

#### Notes:



## Lab activity:

### Monitoring as a Service Step 5 of 7

### Define Management Zones in Dynatrace to create Access Control



In this lab you'll learn how to create Management Zones in Dynatrace to give each team access to the data they are supposed to see.

You will create management zones for:

- The Frontend Team
- The Dev Team
- The Architect Team
- The Operations Team
- The Business Team

### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [06\\_Monitoring\\_as\\_a\\_Service/05\\_Define\\_Management\\_Zones](#)

### Additional Learning

[Management Zones](#)

### Notes :



In this lab you'll see how automation works for the staging environment. To do this you will:

- Create release branches for all microservices
- Build the new release in Jenkins

### sockshop

[add description](#)

S	W	Name ↓	Last Success	Last Failure	Last Duration	
		<a href="#">carts</a>	23 hr - <a href="#">log</a>	N/A	1.3 sec	
		<a href="#">carts.performance</a>	28 days - <a href="#">log</a>	N/A	1.8 sec	
		<a href="#">catalogue</a>	1 mo 5 days - <a href="#">log</a>	N/A	2 sec	
		<a href="#">create-release-branch</a>	26 min - <a href="#">#47</a>	1 day 23 hr - <a href="#">#37</a>	1 min 29 sec	
		<a href="#">front-end</a>	19 hr - <a href="#">log</a>	N/A	1 sec	
		<a href="#">orders</a>	22 hr - <a href="#">log</a>	N/A	1 sec	
		<a href="#">payment</a>	1 mo 5 days - <a href="#">log</a>	N/A	0.82 sec	
		<a href="#">queue-master</a>	23 days - <a href="#">log</a>	N/A	1 sec	
		<a href="#">shipping</a>	23 days - <a href="#">log</a>	N/A	0.83 sec	
		<a href="#">user</a>	23 days - <a href="#">log</a>	N/A	1.1 sec	
		<a href="#">user.performance.multi</a>	28 days - <a href="#">log</a>	N/A	1.4 sec	

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

### Pipeline create-release-branch

This build requires parameters:

SERVICE

Name of the service

[Build](#)

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/06\\_Monitoring\\_as\\_a\\_Service/06\\_Deploy\\_Sockshop\\_to\\_Staging](#)

### Notes :



## Lab activity:

### Monitoring as a Service Step 7 of 7

### Setup Alerting Profile for Availability Issues and Integrate with a ChatOps Tool



In this lab you'll learn how to define an alerting profile to notifying someone when a particular problem is identified by Dynatrace. To do this you will:

Define in Dynatrace:

- An Alerting Profile
- Problem Notification

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository/* [06\\_Monitoring\\_as\\_a\\_Service/07\\_Setup\\_Notification\\_Integration](#)

#### Notes:

## Summary Statements

Can you complete the following summary statements about monitoring as a service?

- The goal of \_\_\_\_\_ is to move from a long manual process of configuring monitoring for a new app to fully automated monitoring.
- Deploying \_\_\_\_\_ achieves this, even in a containerized environment.
- \_\_\_\_\_ are clusters of related processes which run on the same software.
- A \_\_\_\_\_ is a process group running on a host.
- Dynatrace has out of the box and custom \_\_\_\_\_.

## Questions

Do you have any questions about monitoring as a service? Jot them down here and share them with your instructor.

## Performance as a Service

In this module, you will learn more about Performance as a Self Service and work through load testing integration use cases.

Performance as a Self-Service aims on moving

- from manual sporadic execution and analysis of performance tests
- to a fully automated on-demand self-service model for performance testing in a development environment.

Provides early performance feedback which results in better performing software that gets deployed into staging or production environments.

### Use Cases – Key Takeaways

Push Testing Metrics to Dynatrace
Pass Test Script Context via Request Attributes
Automate Analysis
Compare Hotspots

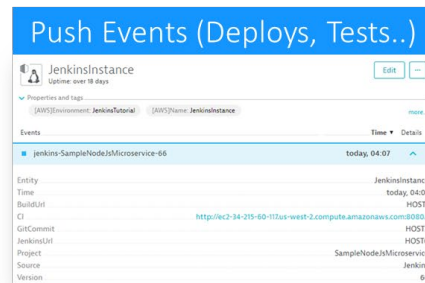
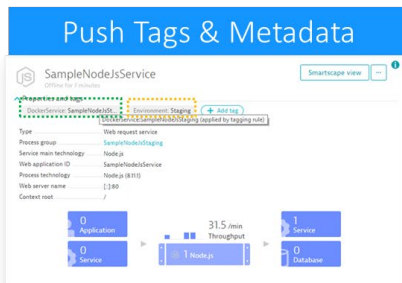
### Load Testing Integration:

Taken from:

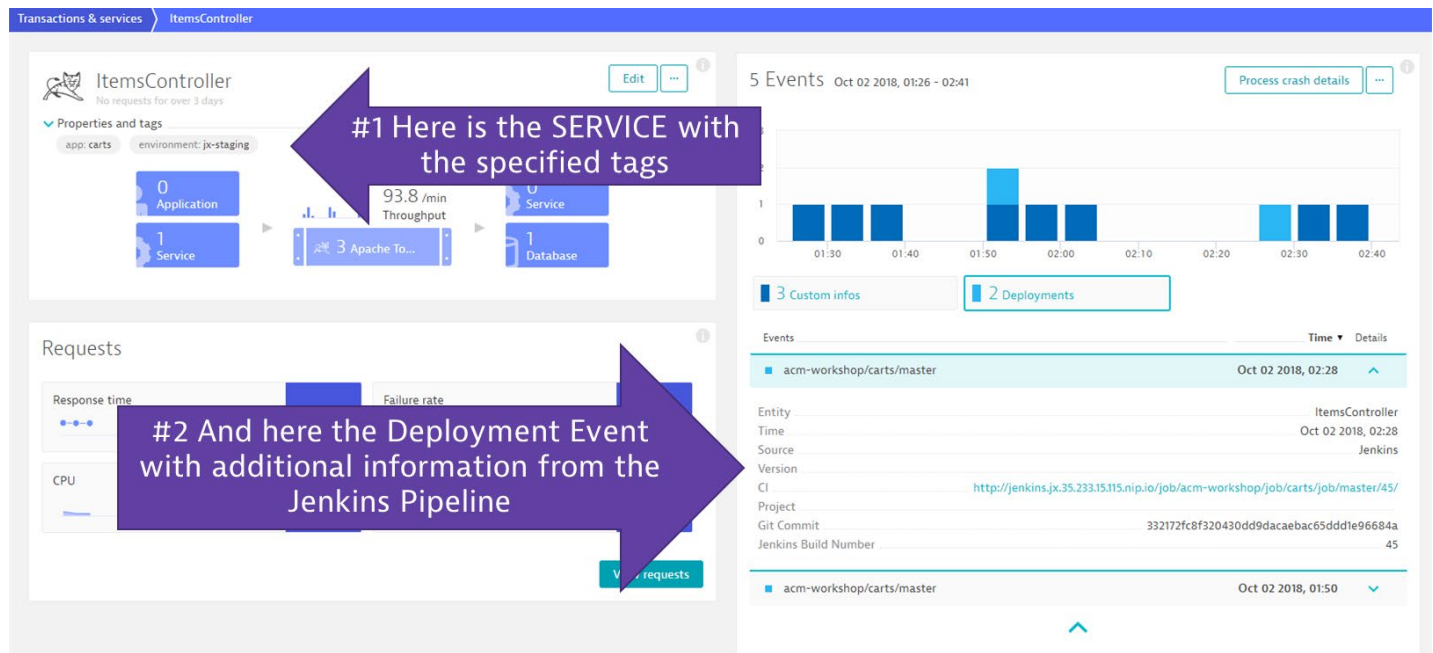
<https://github.com/dynatrace-innovationlab/jenkins-dynatrace-pipeline-tutorial>

How do Jenkins and Dynatrace work together?

	Checkout	Build	CleanStaging	DeployStaging	Testing	ValidateStaging	DeployProduction	WarmUpProduction	ValidateProduction
Average stage times: (Average full run time: ~3min 33s)	1s	2s	2s	2s	1min 56s	1s	3s	3min 47s	2s
#64 Apr 18 03:18 No Changes	1s	4s	1s	2s	2min 8s	2s	2s	1min 9s	1s



## Ensure Proper Tagging and Deployment Events





## Lab activity:

### Performance as a Service Step 1 of 7

### Write Load Test Script

The goal of this lab activity is to learn how to write a JMeter load test that stresses a service with a bulk of requests. To do this you will:

- Add the following parameters:
  - SERVER\_URL - The domain of the service.
  - SERVER\_PORT - The port of the service.
  - CHECK\_PATH - The endpoint to send the requests to.
  - DT\_LTN - The Load Test Name uniquely identifies a test execution.
  - VUCount - The number of virtual users.
  - LoopCount - The number of loops each virtual user performs.
- Correctly tag each request for identification:
  - VU - Virtual User ID of the unique user who sent the request.
  - TSN - Test Step Name is a logical test step within your load testing script (for example, Login or Add to cart).
  - LSN - Load Script Name - name of the load testing script. This groups a set of test steps that make up a multi-step transaction (for example, an online purchase).
  - LTN - The Load Test Name uniquely identifies a test execution (for example, 6h Load Test – June 25)

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository07\\_Performance\\_as\\_a\\_Self\\_Service/01\\_Write\\_Load\\_Test\\_Script](#)

### Additional Learning

<https://www.dynatrace.com/support/help/integrations/third-party/test-automation/dynatrace-and-load-testing-tools-integration/>

### Notes:



## Lab activity:

### Performance as a Service Step 2 of 7

### Define Request Attributes



The goal of this lab activity is to learn how to capture request attributes in Dynatrace based on web request data. To do this you will:

- Create request attributes for Load Test Name (LTN)
- Create request attribute for Test Script Name (TSN)

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository07\\_Performance\\_as\\_a\\_Self\\_Service/02\\_Define\\_Request\\_Attributes](#)

#### Additional Learning

<https://www.dynatrace.com/support/help/monitor/transactions-and-services/request-attributes/how-do-i-capture-request-attributes-based-on-web-request-data/>

#### Notes:



The goal of this lab activity is to learn how to validate the performance of a service. To do this you will:

- Validate average response time
- Validate percentile of response time
- Validate the number of requests and requests/minute
- Server-side failure rate

## Lab activity:

### Performance as a Service Step 3 of 7

### Define Performance Signature for Cart Service

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository07\\_Performance\\_as\\_a\\_Self\\_Service/03\\_Define\\_Performance\\_Signature](#)

#### Additional Learning

<https://www.dynatrace.com/support/help/integrations/third-party/test-automation/dynatrace-and-load-testing-tools-integration/>

#### Notes:





## Lab activity:

### Performance as a Service Step 4 of 7

#### Define Performance Pipeline



The goal of this lab activity is to learn how to build a Jenkins pipeline for implementing the Performance as a Self-Service approach for the carts service used to manually trigger performance testing against a service.. To do this you will:

- Record Dynatrace Session and Push Info Events
- Trigger JMeter Test by a separate function
- Validate the performance signature definition
- Create a performance pipeline for carts

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository* [07\\_Performance\\_as\\_a\\_Self\\_Service/](#)

[04\\_Define\\_Performance\\_Pipeline](#)

#### Notes:



## Lab activity:

### Performance as a Service Step 5 of 7

### Run Performance Tests



The goal of this lab activity is to learn how to run performance testing on current and new implementation of carts. To do this you will:

- Run performance test on current implementation
- Introduce a slowdown in the carts service
- Build the new version
- Run performance testing on the new version
- Explore results in Jenkins

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository* [07\\_Performance\\_as\\_a\\_Self\\_Service/05\\_Run\\_Performance\\_Tests](#)

#### Notes:



## Lab activity:

### Performance as a Service Step 6 of 7

### Compare Tests in Dynatrace



The goal of this lab activity is to learn how to leverage Dynatrace to identify the difference between two performance tests. To do this you will:

- Open Dynatrace from Jenkins Pipeline
- Narrow down the requests based on request attributes
- Open the Comparison view
- Compare response time hotspots

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[07\\_Performance\\_as\\_a\\_Self\\_Service/06\\_Compare\\_Tests\\_in\\_Dynatrace](#)

#### Notes:



## Lab activity:

### Performance as a Service Step 7 of 7

### Retry Performance Test



The goal of this lab activity is to remove the slowdown in the carts service to have a solid version.. To do this you will:

- Revert the original behavior of carts
- Build the new version
- Run performance test on the new version.

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[07\\_Performance\\_as\\_a\\_Self\\_Service/07\\_Retry\\_Performance\\_Test](#)

#### Notes:

## Summary

- Performance as a Self-Service aims on moving
  - from manual sporadic execution and analysis of performance tests
  - to a fully automated on-demand self-service model for performance testing in a development environment.
- Provides early performance feedback which results in better performing software that gets deployed into staging or production environments.
- Load Testing Integration
  - Push Testing Metrics to Dynatrace
  - Pass Test Script Context via Request Attributes
  - Automate Analysis: Compare Builds and Method Hotspots in Dynatrace
- Performance as a Self-Service
  - Fully automated on-demand self-service model for performance testing in a development environment

## Questions

Do you have any questions about Performance as a Self-Service? Jot them down here and share them with your instructor.

## Production Deployments

In this module, we will describe production deployments, deployment strategies, and showcase using Istio on Kubernetes to canary-deploy a new front-end version.

### Deployment Strategies

<p><b>Recreate</b></p> <p>Shut down version A</p> <p>Deploy version B after A is turned off</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"> <li>• Easy to do</li> <li>• Application state entirely renewed</li> </ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"> <li>• Downtime – High impact on the user</li> </ul>
<p><b>Ramped</b></p> <p>aka Rolling upgrade</p> <p>Replace instances with new version one by one</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"> <li>• Easy to setup</li> <li>• New version is slowly released</li> <li>• Stateful applications can rebalance data</li> </ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"> <li>• No control over traffic</li> </ul>
<p><b>Blue/Green</b></p> <p>Version B (green) is deployed alongside version A (blue)</p> <p>Traffic is switched to green at load balancer</p> <p>Blue is not immediately deleted</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"> <li>• Instant rollout and rollback</li> <li>• Avoiding version conflicts</li> </ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"> <li>• Difficult for stateful applications</li> <li>• Long running transactions</li> </ul>

<p><b>Canary</b></p> <p>Gradually shifting traffic from version A to B</p> <p>E.g.: 90% A - 10% B for 10 mins, then 80% A - 20% B ...</p> <p>Controlled rollout with easy rollback</p> <p>Criteria for traffic distribution</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"><li>• New version only released to a subset of users</li><li>• Tryouts under production conditions</li><li>• Fast and easy rollback</li></ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"><li>• Difficult for stateful applications</li></ul>
<p><b>A/B Testing</b></p> <p>No deployment strategy but related</p> <p>Testing conversion rates of features</p> <p>Criteria for traffic distribution</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"><li>• Run several versions in parallel</li><li>• Full traffic control</li></ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"><li>• Requires intelligent load balancer (L7)</li><li>• Hard to troubleshoot</li></ul>
<p><b>Shadow</b></p> <p>Deploy version B alongside version A</p> <p>Mirror traffic to version B</p> <p>Go live when quality requirements are met</p>	
<p><b>Pros:</b></p> <ul style="list-style-type: none"><li>• Testing B under real conditions</li><li>• No user impacts</li><li>• Controlled rollout</li></ul>	<p><b>Cons:</b></p> <ul style="list-style-type: none"><li>• Expensive – twice the resources</li><li>• Not a true user test</li><li>• May require mocking services</li></ul>

## Istio architecture

Istio is an open source project initiated by Google and IBM used to manage service interactions.

### Pilot

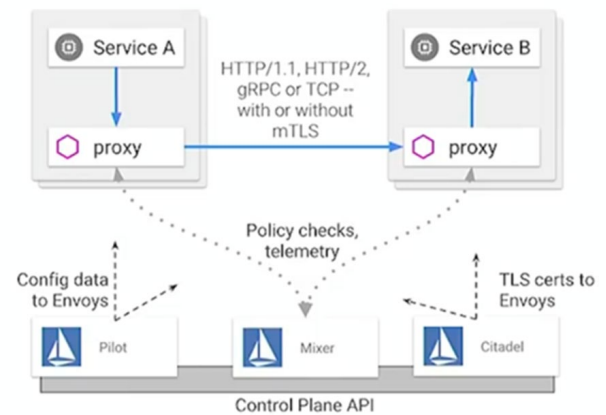
- Control plane to configure and push service communication policies
- Routing and forwarding policies

### Mixer

- Policy enforcement with a flexible plugin model for providers for a policy

### Citadel

- Service-to-service authentication and encryption using mutual TLS with built-in identify and credential management



## Istio components

Gateway	<pre> 1  apiVersion: networking.istio.io/v1alpha3 2  kind: Gateway 3  metadata: 4    name: sockshop-gateway 5  spec: 6    selector: 7      istio: ingressgateway 8    servers: 9      - port: 10         number: 80 11         name: http 12         protocol: HTTP 13       hosts: 14         - "*" </pre>
VirtualService	<pre> 1  apiVersion: networking.istio.io/v1alpha3 2  kind: VirtualService 3  metadata: 4    name: sockshop 5  spec: 6    hosts: 7      - "*" 8    gateways: 9      - sockshop-gateway 10   http: 11     - route: 12       - destination: 13         host: front-end.production.svc.cluster.local 14         subset: v1 </pre>



DestinationRule	<pre>1  apiVersion: networking.istio.io/v1alpha3 2  kind: DestinationRule 3  metadata: 4    name: front-end-destination 5  spec: 6    host: front-end.production.svc.cluster.local 7    subsets: 8      - name: v1 9        labels: 10         version: v1 11      - name: v2 12        labels: 13         version: v2</pre>
ServiceEntry	<pre>1  apiVersion: networking.istio.io/v1alpha3 2  kind: ServiceEntry 3  metadata: 4    name: carts-and-orders-db 5  spec: 6    hosts: 7      - "*.mongodb.net" 8    addresses: 9      - 18.214.93.113 10     - 18.215.19.51 11     - 54.82.134.106 12    ports: 13      - number: 27017 14        name: tcp 15        protocol: TCP 16    location: MESH_EXTERNAL</pre>



## Lab activity:

### Production Deployment Step 1 of 6

### Install Istio



The goal of this lab activity is to install Istio components and verify that they are running in the k8s cluster To do this you will:

- Go to your home directory on the bastion host
- Install Istio's custom resource definitions (CRDs)

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[08\\_Production\\_Deployments/08\\_Production\\_Deployments/1\\_Install\\_istio](#)

#### Notes:



## Lab activity:

### Production Deployment Step 2 of 6

### Configure Istio Components



The goal of this lab activity is to enable Istio's automatic sidecar injection for one k8s namespace. To do this you will:

- Create and configure mandatory Istio components
- Allow external traffic to leave the service mesh

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository08\\_Production\\_Deployments/2\\_Configure\\_istio\\_components](#)

### Notes:



## Lab activity:

**Deploy to  
Production  
Step 3 of 6**

**Deploy to  
Production**



The goal of this lab activity is to promote all components that are currently in the staging name space to the production namespace. To do this you will:

- Trigger the pipeline in Jenkins

### Instructions

This lab is found in the GitHub workshop repository:

[course-repositor08\\_Production\\_Deployments/03\\_Define\\_Performance\\_Signature](#)

### Notes:



## Lab activity:

### Production Deployment Step 4 of 6

#### Create front-end v2



The goal of this lab activity is to create an improved version of the front-end service. To do this you will:

- Change the color of the application header
- View the effect of traffic routing between two different artifact versions

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository* [08\\_Production\\_Deployments/4\\_Create\\_front-end\\_v2](#)

#### Notes:



## Lab activity:

### Production Deployment Step 5 of 6

### Deploy front-end v2



The goal of this lab activity is to promote the new version of the front-end service to production. To do this you will:

- Trigger the parameterized pipeline
- Enter parameters

#### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[08\\_Production\\_Deployments/5\\_Deploy\\_front-end\\_v2](#)

#### Notes:



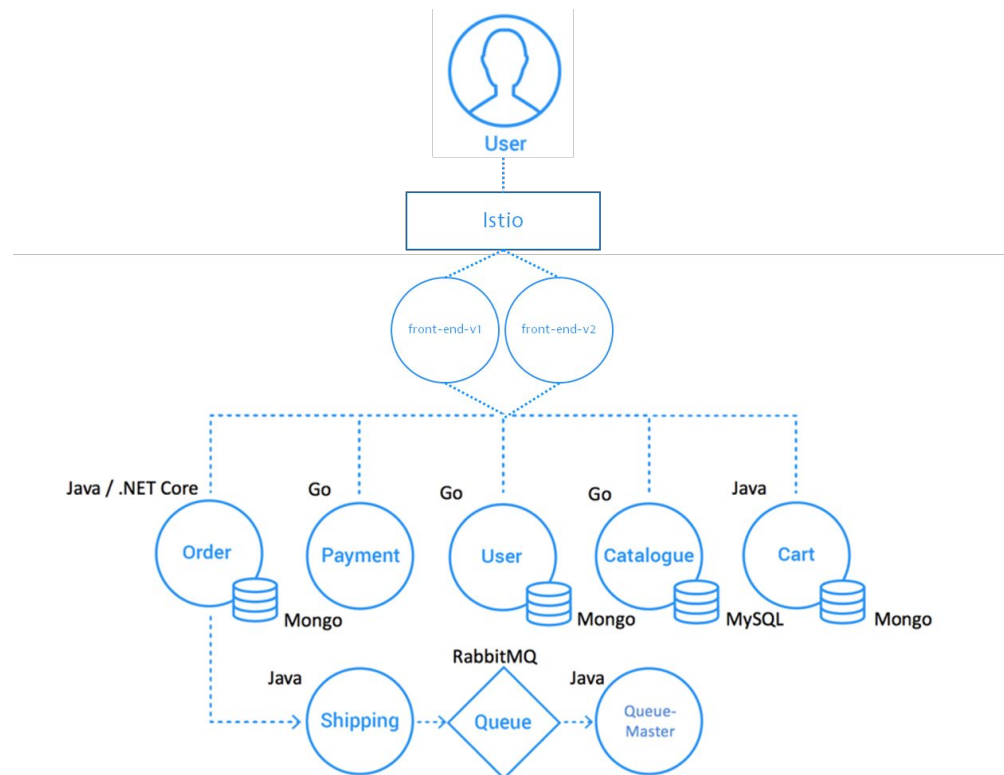
## Lab activity:

### Production Deployment Step 6 of 6

### Istio Traffic Routing

The goal of this lab activity is to configure traffic routing in Istio to redirect traffic based on different criteria. To do this you will:

- Redirect traffic using weight rules
- Redirect logged in users
- Redirect Chrome users



### Instructions

This lab is found in the GitHub workshop repository:

*course-repository* [6\\_Istio\\_Traffic\\_Routing](#)

### Notes:

## Summary

- Production deployments from idea to production
- Deployment strategies
  - Recreate - Shut down version A; Deploy version B after A is turned off
  - Ramped - aka Rolling upgrade; Replace instances with new version one by one
  - Blue/Green - Version B (green) is deployed alongside version A (blue); Traffic is switched to green at load balancer; Blue is not immediately deleted
  - Canary - Gradually shifting traffic from version A to B; Controls rollout with easy rollback; Criteria for traffic distribution
  - A/B testing - No deployment strategy but related; Tests conversion rates of features; Criteria for traffic distribution
  - Shadow - Deploy version B alongside version A; Mirror traffic to version B; Go live when quality requirements are met
- Istio
  - Open source project initiated by Google and IBM
  - Reviewed architecture and components

## Questions

Do you have any questions about Deployment Strategies or the steps taken in the labs? Jot them down here and share them with your instructor.



## Runbook Automation and Self-Healing

In this module, you will learn about Ansible and Ansible tower.

### Runbooks:

- “compilation of routine procedures and operations that the system operator (administrator) carries out”

### Ansible:

- Simple automation language + automation engine
- Human readable (yaml)

### Ansible Tower:

- Web UI
- Management of runbooks
- Powerful API for automation



**CONFIG MANAGEMENT**



**APP DEPLOYMENT**



**PROVISIONING**



**CONTINUOUS DELIVERY**



**SECURITY & COMPLIANCE**



**ORCHESTRATION**



## Lab activity:

### Runbook Automation Step 1 of 4

### Deploy Ansible Tower



The goal of this lab activity is to deploy Ansible Tower for automation inside of our cluster. To do this you will:

- Go to <https://www.ansible.com/license>
- Setup a free trial license file.
- Deploy Ansible Tower

### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[09\\_Runbook\\_Automation\\_and\\_Self\\_Healing/01\\_Deploy\\_Ansible\\_Tower](#)

### Notes:



## Lab activity:

### Runbook Automation Step 2 of 4

### Setup Tower

The goal of this lab activity is to setup and configure our Ansible Tower environment. To do this you will:

- Add Github credentials to be able to check out Github repository
- Create a project in Ansible Tower that holds defines which repository to use
- Create an inventory that holds additional information such as userdata and variables
- Create job templates that can then be executed and will run our playbooks.

### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[09\\_Runbook\\_Automation\\_and\\_Self\\_Healing/02\\_Setup\\_Tower](#)

### Notes:



## Lab activity:

### Runbook Automation Step 3 of 4

### Setup Dynatrace



The goal of this lab activity is to integrate Ansible Tower Runbook in Dynatrace. To do this you will:

- Setup Problem Notifications
- Login to your Ansible Tower Instance

### Instructions

This lab is found in the GitHub workshop repository:

*course-repository*[09\\_Runbook\\_Automation\\_and\\_Self\\_Healing/03\\_Setup\\_Dynatrace](#)

### Notes:



## Lab activity:

### Runbook Automation Step 4 of 4

### Run Playbook



The goal of this lab activity is run our promotional campaign in our production environment by applying a change to our configuration of the carts service. To do this you will:

- Run the campaign in our production environment

### Instructions

This lab is found in the GitHub workshop repository:

[course-repository09\\_Runbook\\_Automation\\_and\\_Self\\_Healing/04\\_Run\\_Playbook](#)

### Notes:

## Summary Statements

Can you complete the following summary statements about runbook automation?

- \_\_\_\_\_ are a compilation of routine procedures and operations that the system operator (administrator) carries out.
- \_\_\_\_\_ is a simple automation language + automation engine.

## Questions

Do you have any questions about Runbook Automation or the steps taken in the labs? Jot them down here and share them with your instructor.

## Unbreakable Delivery Pipeline

The overall goal of the Unbreakable Delivery Pipeline is to implement a pipeline that prevents bad code changes from impacting your real end users. Therefore, it relies on three concepts known as Shift-Left, Shift-Right, and Self-Healing:

**Shift-Left:** Ability to pull data for specific entities (processes, services, applications) through an Automation API and feed it into the tools that are used to decide on whether to stop the pipeline or keep it running.

**Shift-Right:** Ability to push deployment information and meta data to your monitoring environment, e.g.: differentiate BLUE vs GREEN deployments, push build or revision number of deployment, notify about configuration changes.

**Self-Healing:** Ability for smart auto-remediation that addresses the root cause of a problem and not the symptom

In this module you will learn how to implement such an Unbreakable Delivery Pipeline with Dynatrace and Ansible Tower.

### Hands-on Building the Unbreakable Delivery Pipeline

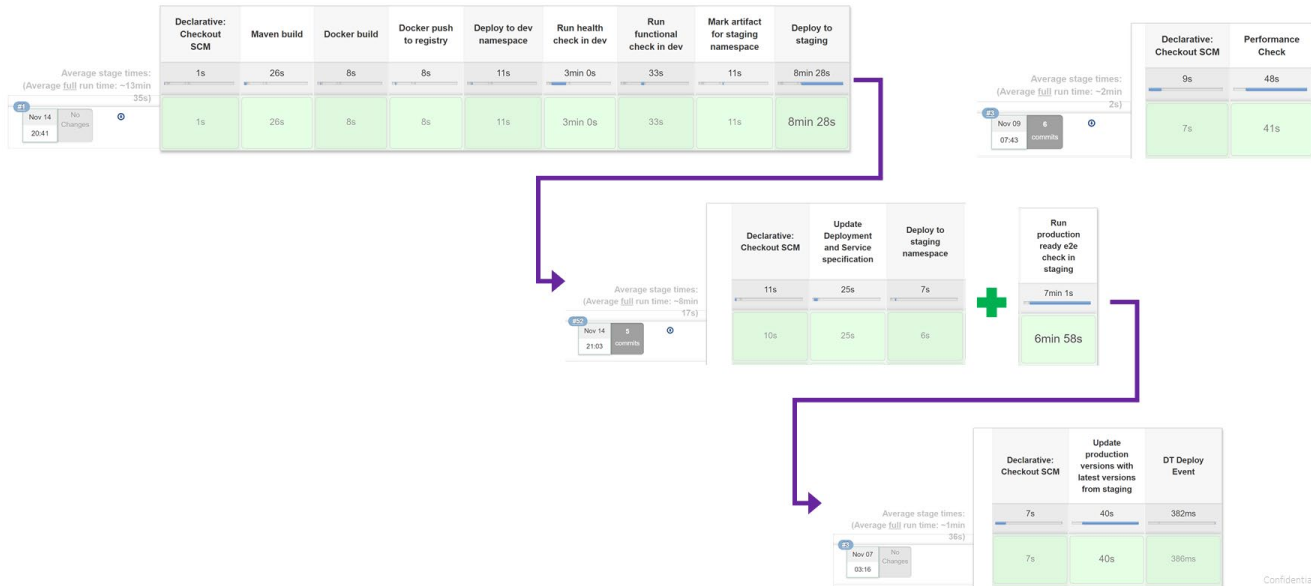


#### Current State





## Shift-Left: Break Pipeline Earlier



Confidential

18

## Additional Learning

For more information about Unbreakable Pipeline read:

<https://www.dynatrace.com/news/blog/unbreakable-devops-pipeline-shift-left-shift-right-self-healing/>





## Lab activity:

### Unbreakable Delivery Pipeline

#### Prep Work

In this module you will learn how to implement an Unbreakable Delivery Pipeline with Dynatrace and Ansible Tower by following these four labs:

- Harden Staging Pipeline with Quality Gate
- Simulate Early Pipeline Break
- Setup Self-Healing for Production
- Introduce a Failure into Front-End
- Simulate a Bad Production Deployment

#### Instructions

First, there are a few steps to perform:

1. Add Env Variables to Jenkins
2. Add a Container to the Kubernetes Pod Template kubegit

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/00\\_Preparation\\_for\\_Unbreakable\\_Delivery\\_Pipeline](#)

#### Notes:



## Lab activity:

### Unbreakable Delivery Pipeline

#### Part 1 of 5

### Harden Staging Pipeline with Quality Gate



In this lab activity you'll add an additional quality gate to your CI pipeline. An end-to-end check will verify the functionality of the Sockshop application in the staging environment. To do this you will perform the following steps:

- Comment out the "DT Deploy Event"
- Add e2e Test to Staging Pipeline
- Set the Upper and Lower Limit in the Performance Signature

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/01\\_Harden\\_Staging\\_Pipeline\\_with\\_Quality\\_Gate](#)

#### Notes:



## Lab activity:

### Unbreakable Delivery Pipeline

#### Part 2 of 5

### Simulate Early Pipeline Break

In this lab you'll release a service to staging that is not tested based on performance tests. Intentionally, the service is slowed down to fail at the e2e check in the staging pipeline. To do this you will perform the following steps:

- Introduce a Slowdown into the Carts Service
- Create a new Release
- Build the new Release in Jenkins
- Follow the Jenkins Build Pipelines
- Remove the Slowdown in the Carts Service
- Create a new Release
- Build the new Release in Jenkins

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/02\\_Simulate\\_Early\\_Pipeline\\_Break](#)

#### Notes:



## Lab activity:

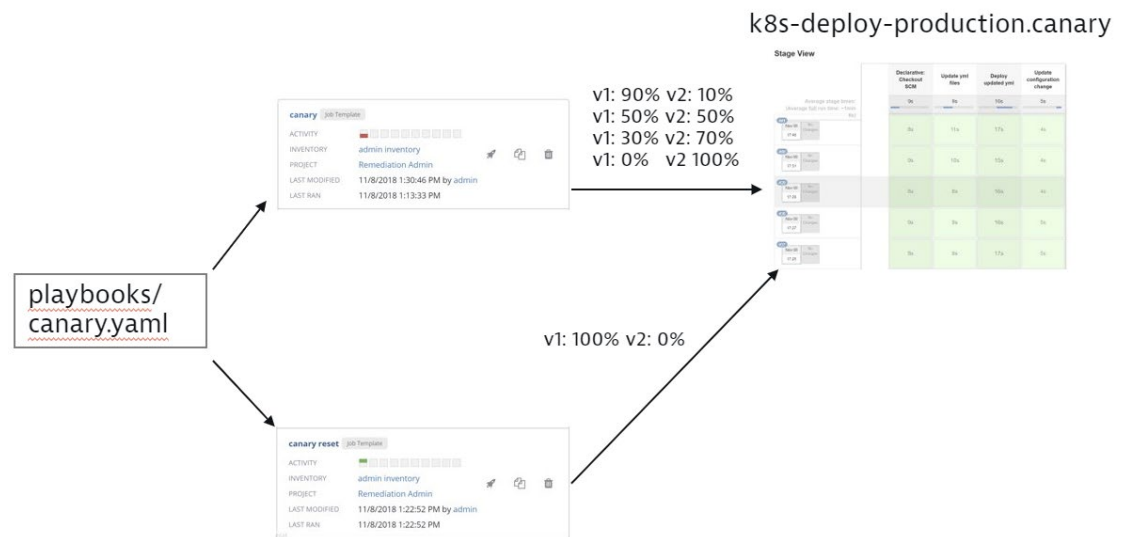
### Unbreakable Delivery Pipeline

#### Part 3 of 5

### Setup Self-Healing Action for Production Deployment

In this lab you'll create an Ansible Tower job that releases a deployment in a calaray release manner. Additionally, you will create a second job that switches back to the old version in case the canary (i.e., the new version of front-end) behaves wrong. To do this you will perform the following steps:

- Create Job Template for Canary Release in Ansible Tower
- Duplicate Job Template for Self-Healing in Ansible Tower



## Instructions

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/03\\_Setup\\_Self\\_Healing\\_for\\_Production](#)

## Notes:



## Lab activity:

### Unbreakable Delivery Pipeline

#### Part 4 of 5

### Introduce a Failure into Front-End



In this lab you will introduce a Java Script error into front-end and deploy it as version 2. To do this perform the following steps:

- Introduce a JavaScript error in the Front-end Service
- Create a new Release
- Build the new Release in Jenkins
- Delete the Virtual Service for Sockshop
- Deploy the new Front-End to Production
- Apply the Virtual Service for Sockshop

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/04\\_Introduce\\_a\\_Failure\\_into\\_Front-End](#)

#### Notes:



## Lab activity:

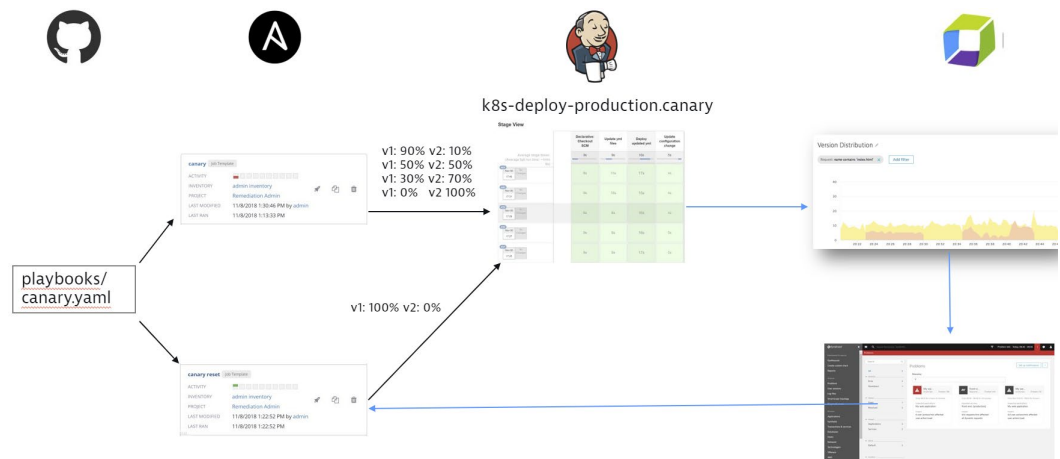
### Unbreakable Delivery Pipeline

#### Part 5 of 5

### Simulate a Bad Production Deployment

In this lab you'll create a deployment of the front-end service that passes the quality gate in the staging pipeline and will be deployed to production. All traffic will be routed to this new version using a prepared deployment pipeline. To do this you will perform the following steps:

- Create a Synthetic Monitor in Dynatrace
- Run Job Template in the Ansible Tower
- Adjust Sensitivity of Anomaly Detection
- Wait for a Problem to Appear in Dynatrace



### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/10\\_Unbreakable\\_Delivery\\_Pipeline/05\\_Simulate\\_a\\_Bad\\_Production\\_Deployment](#)

### Notes:

## Questions

Do you have any questions about Deployment Strategies or the steps taken in the labs? Jot them down here and share them with your instructor.

## Virtual Operations

### Using the Dynatrace API

Use the Dynatrace API to automate your monitoring tasks and export different types of data into your third-party reporting and analysis tools. The Dynatrace API uses a token mechanism for authentication. You need to generate the token and assign the required permissions to it.

<https://www.dynatrace.com/support/help/dynatrace-api/dynatrace-api-authentication/>

Dynatrace API is a powerful HTTP REST API that allows you to seamlessly integrate with Dynatrace to build your own customized tasks.

### Using Dynatrace Davis

When we started with Davis 2 years ago, the hype around natural language processing was just starting. We had two personas in mind:

- Operations teams get a virtual team member, that answers technical questions through a conversational interface and serves as starting point to drill deeper into problems if there are any.
- Executives get a virtual assistant, that answers high level questions about overall application health.

Without the need to log in to the product, Davis blends right into daily routines and all it takes is to ask Davis for a report.

Davis is available for free at: <https://davis.dynatrace.com>





## Lab activity:

### Virtual Operations

#### Part 1 of 4

### Check Prerequisites



In this lab you will explore the Dynatrace API. You will do this by performing the following 4 labs:

- Check Prerequisites
- Explore the Dynatrace API
- Monitor Host Utilization
- Get the Most Volatile Service

#### Instructions

To get started, you will check that Node.js is installed on the Bastion host.

This lab is found in the GitHub workshop repository:

*course-repository*/[11\\_Virtual\\_Operations\\_and\\_API/ 1\\_Check\\_Prerequisites](#)

#### Notes:



## Lab activity:

### Virtual Operations

#### Part 2 of 4

#### Explore the Dynatrace API



Dynatrace provide extensive APIs for querying and manipulating data. All API related information and documentation is located inside your Dynatrace environment under Settings / Integration / Dynatrace API.

To start exploring the Dynatrace API, you will first need to:

- Create an API Token
- Authorize the API Token

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/11\\_Virtual\\_Operations\\_and\\_API/2\\_Explore\\_the\\_Dynatrace\\_API](#)

#### Notes:



## Lab activity:

### Virtual Operations

#### Part 3 of 4

### Monitor Host Utilization



In this lab you will create a simple dashboard that analyzes host utilization and display the host instances that are underutilized.

To do this you will need:

- A list of all hosts
- A way to find out the CPU utilization for a certain timeframe for a given host

During this lab we will walk you through setting this up.

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/11\\_Virtual\\_Operations\\_and\\_API/3\\_Monitor\\_Host\\_Utilization](#)

#### Notes:



## Lab activity:

### Virtual Operations

#### Part 4 of 4

#### Get the Most Volatile Service



The lab activity shows you how to analyze deployment events to create a list of services ranked by the number of deployment events.

To do this you will need:

- A list of all services
- The number of deployment events per service

#### Instructions

This lab is found in the GitHub workshop repository:

[course-repository/11\\_Virtual\\_Operations\\_and\\_API/4\\_Get\\_the\\_Most\\_Volatile\\_Service](#)

#### Notes:

### Summary Statements

- \_\_\_\_\_ is a powerful HTTP REST API that allows you to seamlessly integrate with Dynatrace to build your own customized tasks.
- \_\_\_\_\_ is available to everyone, can be integrated with Alexa and Slack.

### Questions

Do you have any questions about using the Dynatrace API or Davis? Jot them down here and share them with your instructor.

## Autonomous Cloud Lab Questions

You have reached the end of the planned exercises. Do you have any remaining questions? Jot them here and be sure to address them with your instructor.

## ACL Key Takeaways

What key points will you take away from the Autonomous Cloud Lab? Jot them here for reference when you return to home base.

## Contact Information

Colin Lesko  
PDP Team Lead  
Email: [Collin.lesko@dynatrace.com](mailto:Collin.lesko@dynatrace.com)