dynatrace

Autonomous Cloud Lab

Participant Guide

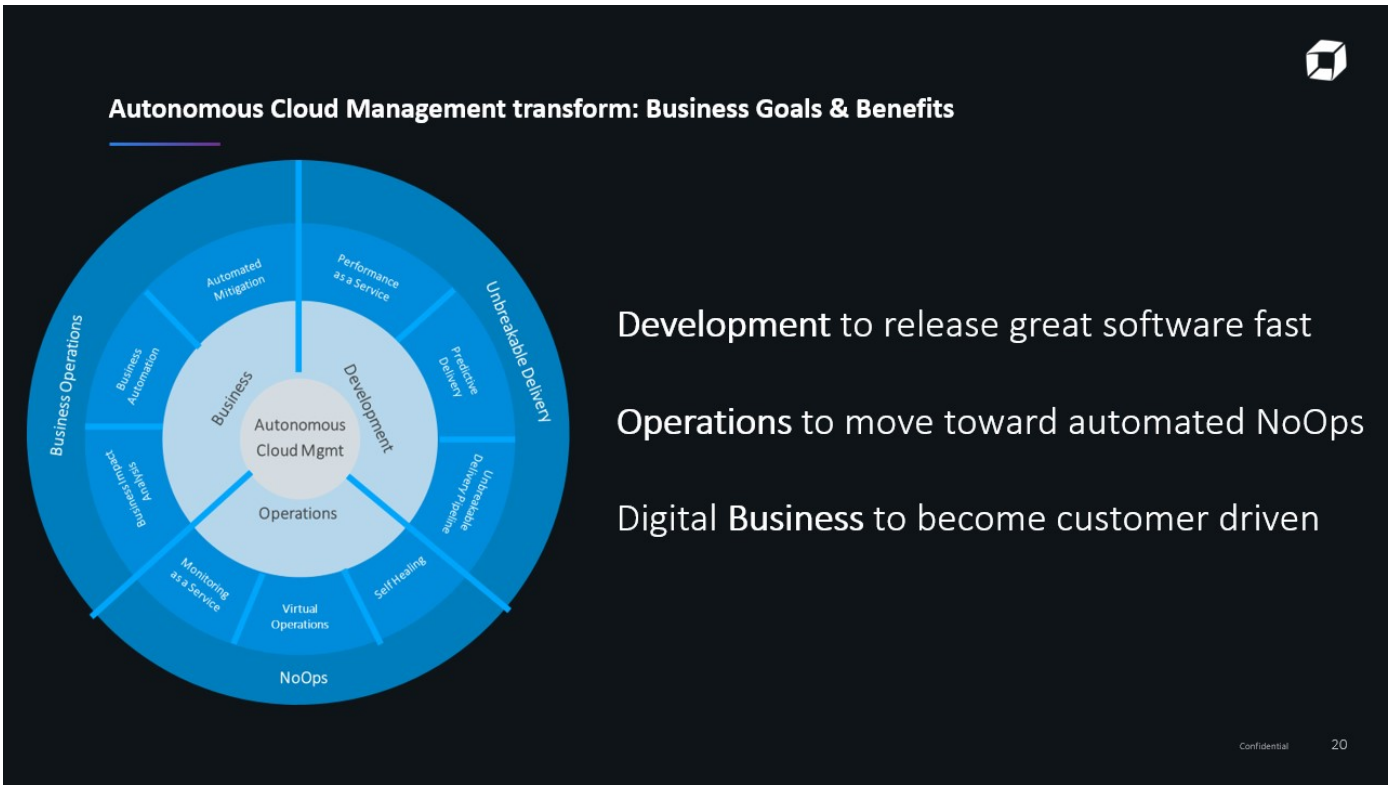
Usernames and Passwords

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

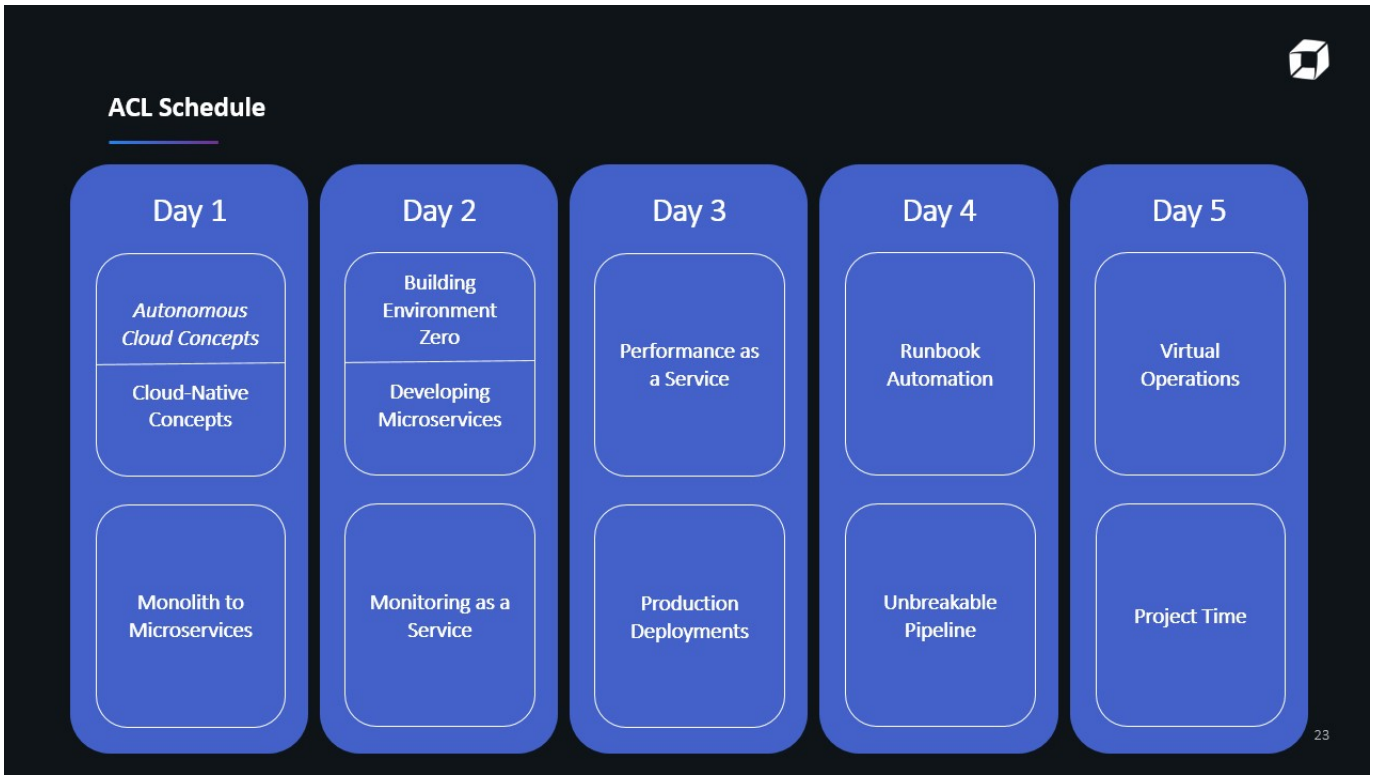
Autonomous Cloud Management



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

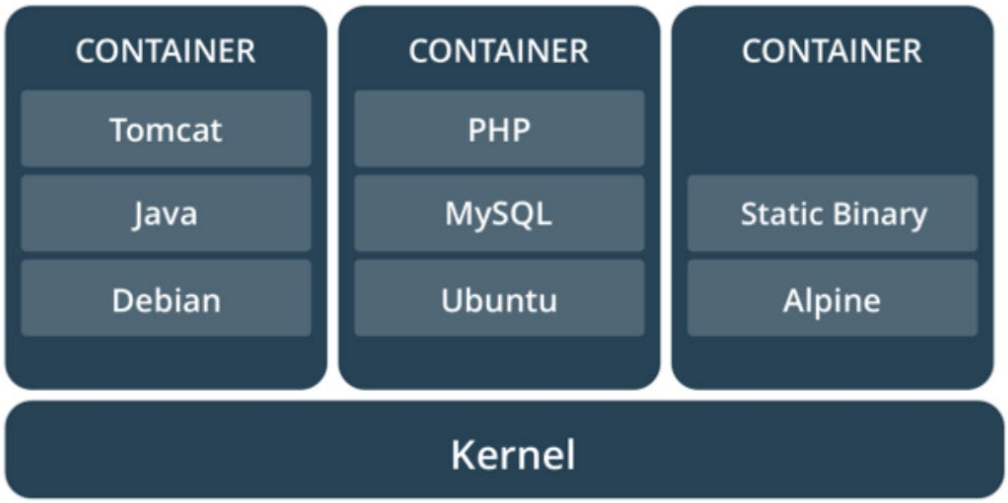
In this module you will learn a definition of Cloud-native concepts such as *Microservice*, *Container (and Container Image)*, *Container Registry*, and *Kubernetes*.

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.

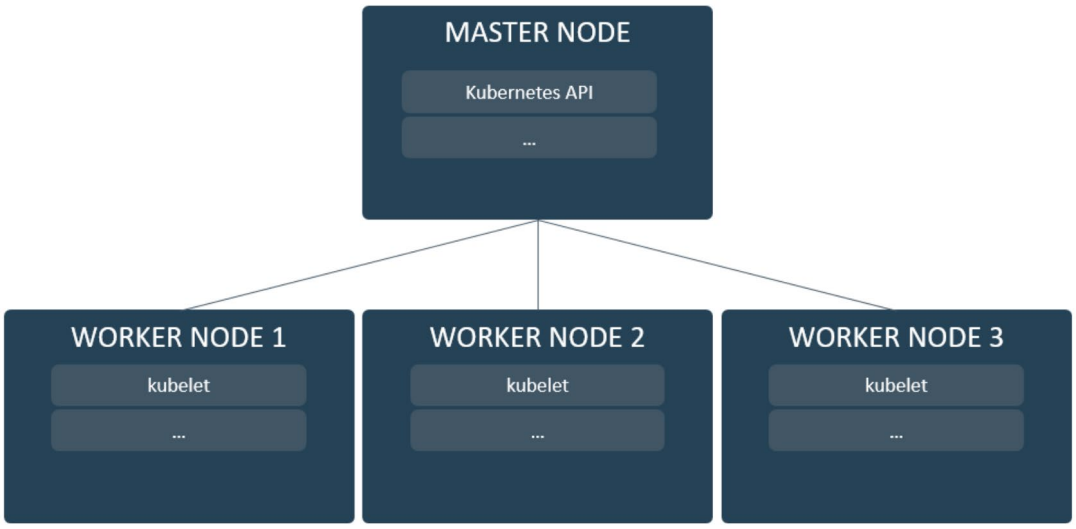
This lab is found in the GitHub workshop repository:

course-repository/02_Cloud_native_Concepts/01_Working_with_Containers/

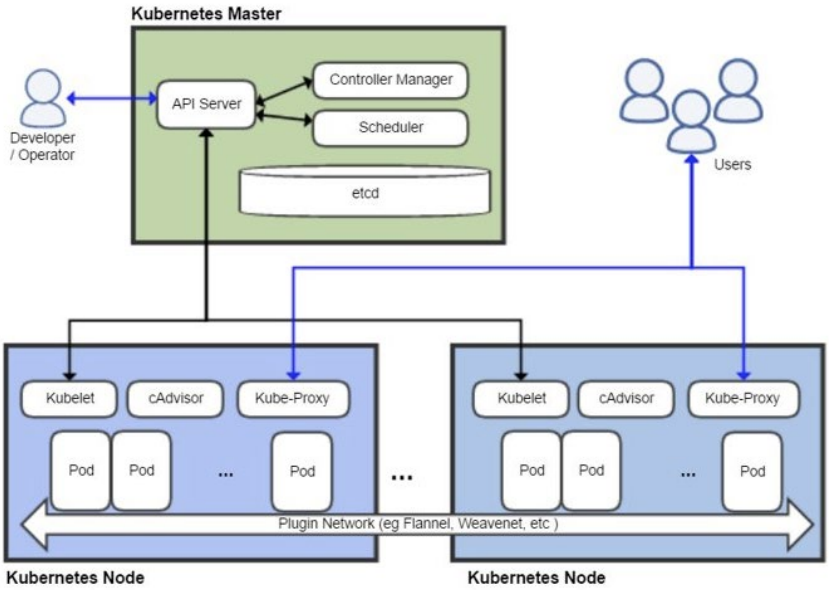
Notes:

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.

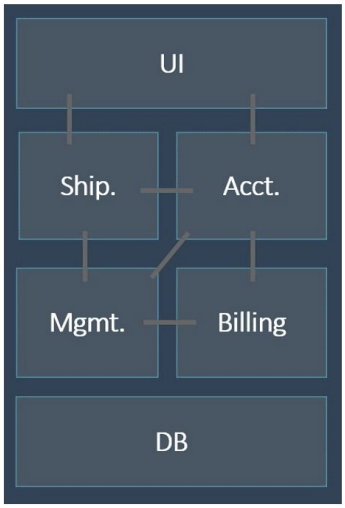
Questions

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 Dynatracer.

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



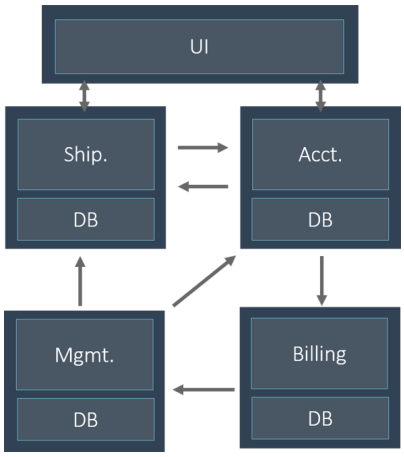
Agility – Rebuilding the whole application takes a decent amount of time

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:

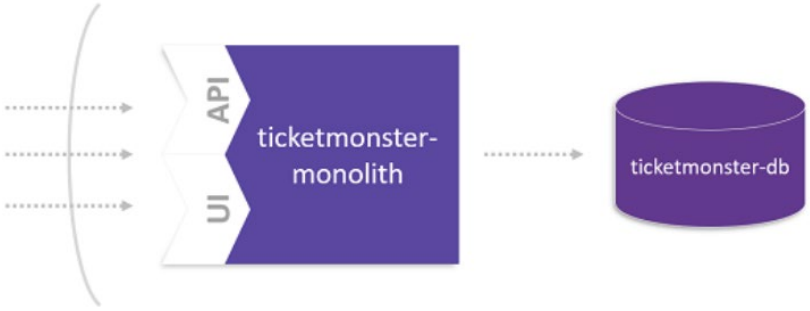
Monolith to Microservices

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:

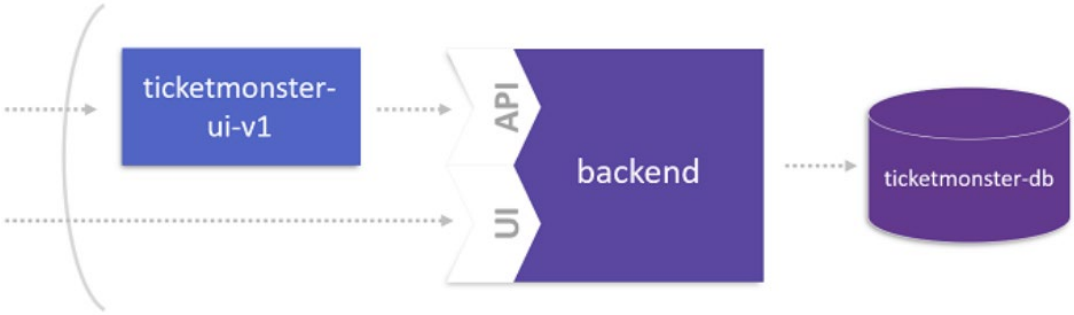
Monolith to Microservices

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:

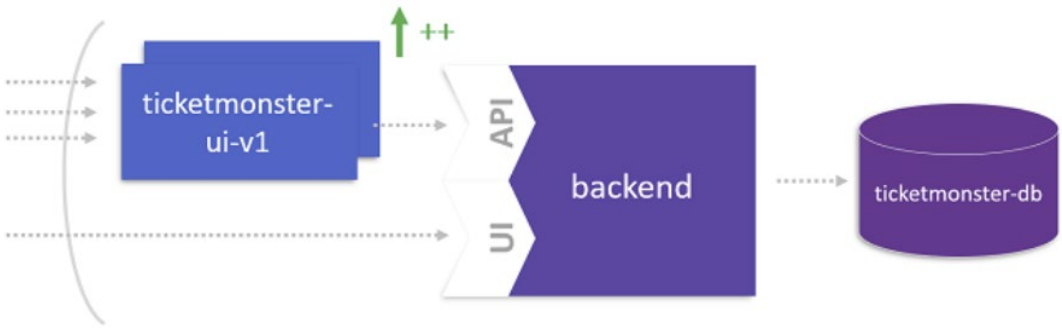
Monolith to Microservices

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:

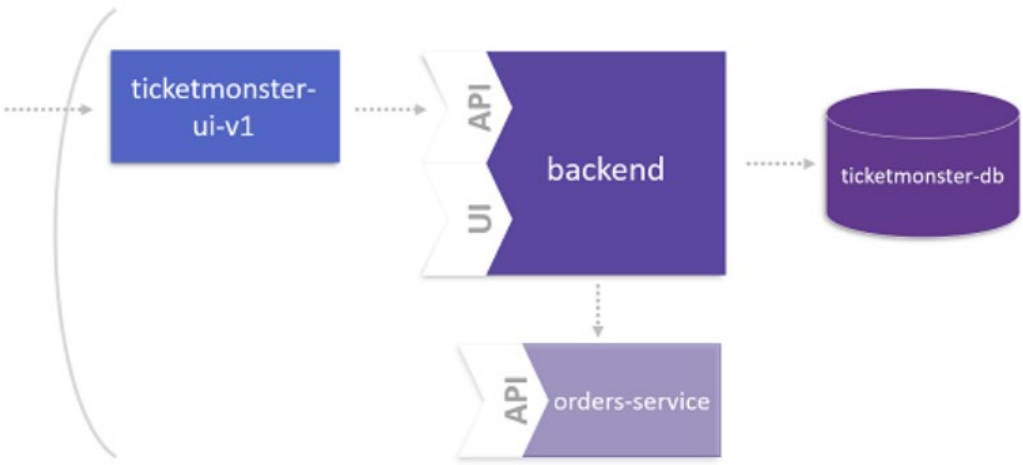
Monolith to Microservices

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

Notes:

Lab activity:

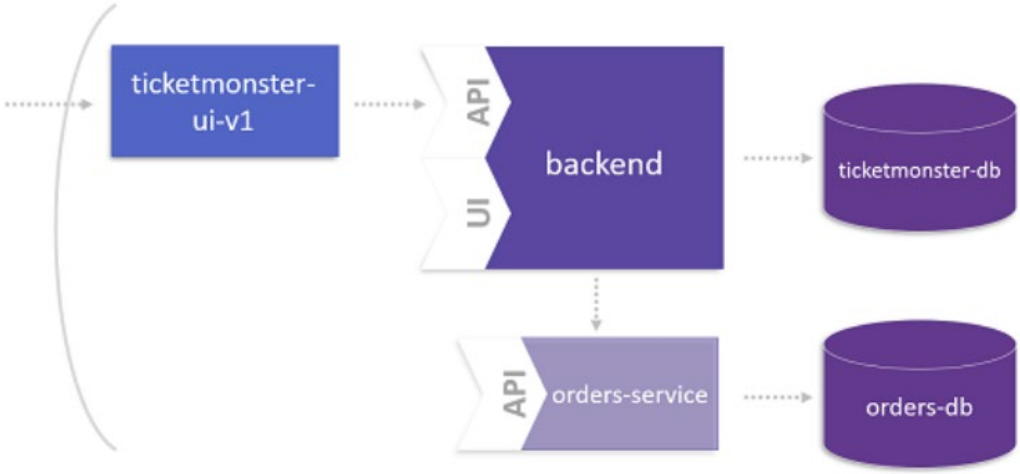
Monolith to Microservices

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

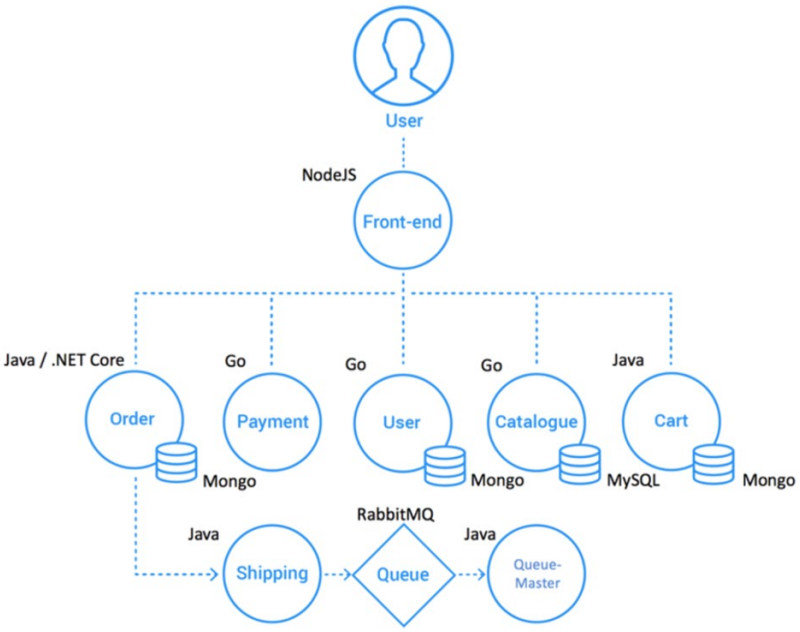
Do you have any questions about moving your monolith application to microservices? Jot them down here and share them with your instructor.

Building Environment Zero

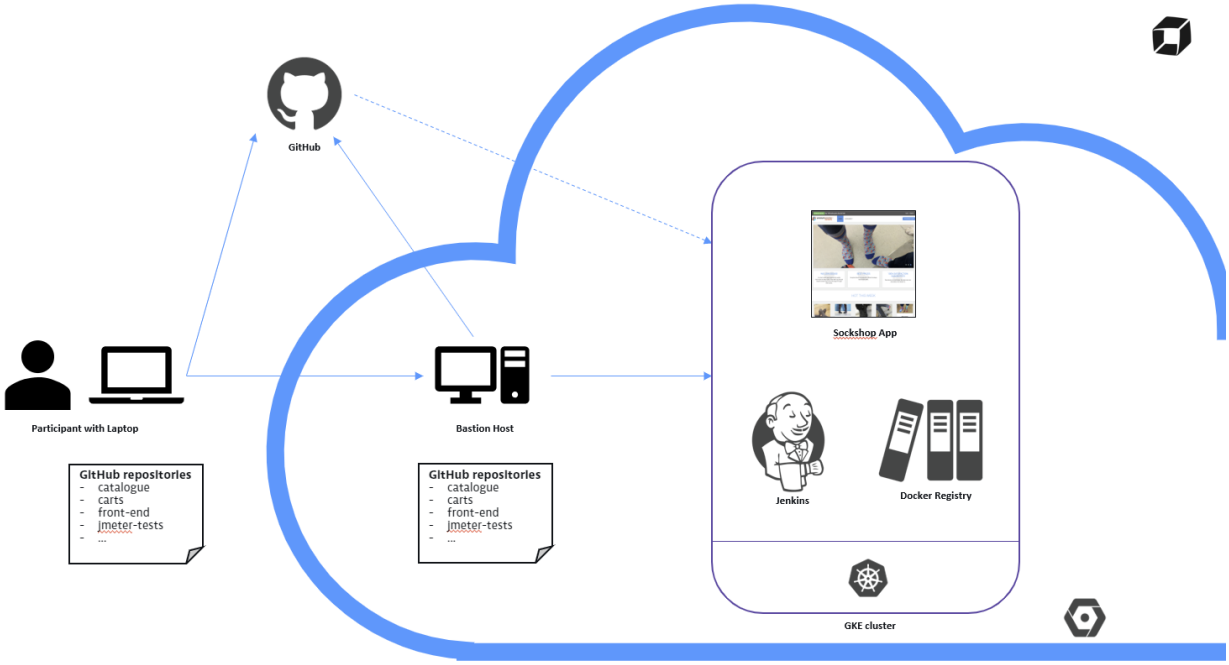
Sockshop Application

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

kubectl Cheat Sheet

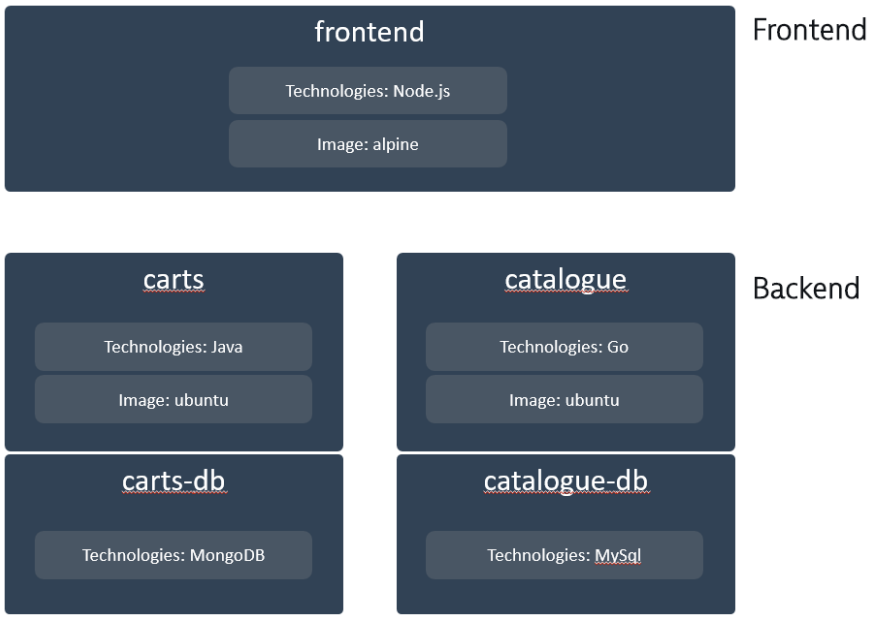
Notes:

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.

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 familier 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

Instructions

This lab is found in the GitHub workshop repository:

course-repository/05_Developing_Microservices/02_Deploy_Microservice_to_Dev

Notes:

Lab activity:

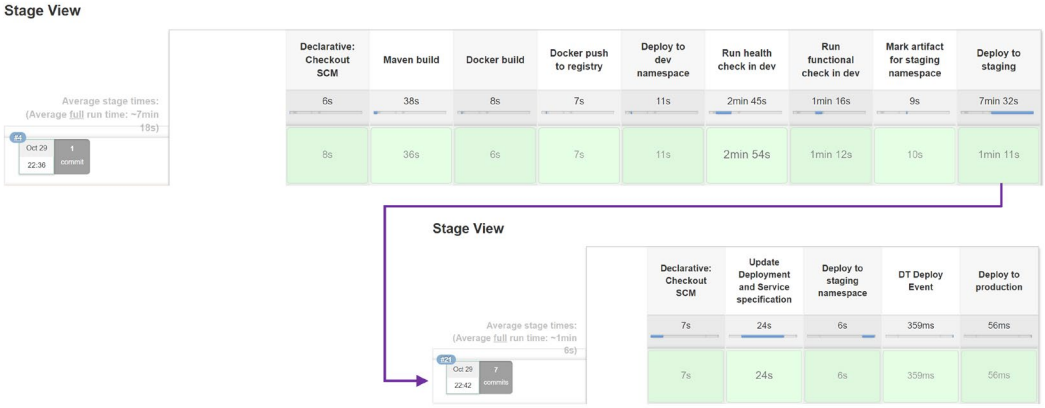
Developing Microservices

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



Instructions

This lab is found in the GitHub workshop repository:

course-repository/05_Developing_Microservices/03_Deploy_Microservice_to_Staging

Notes:

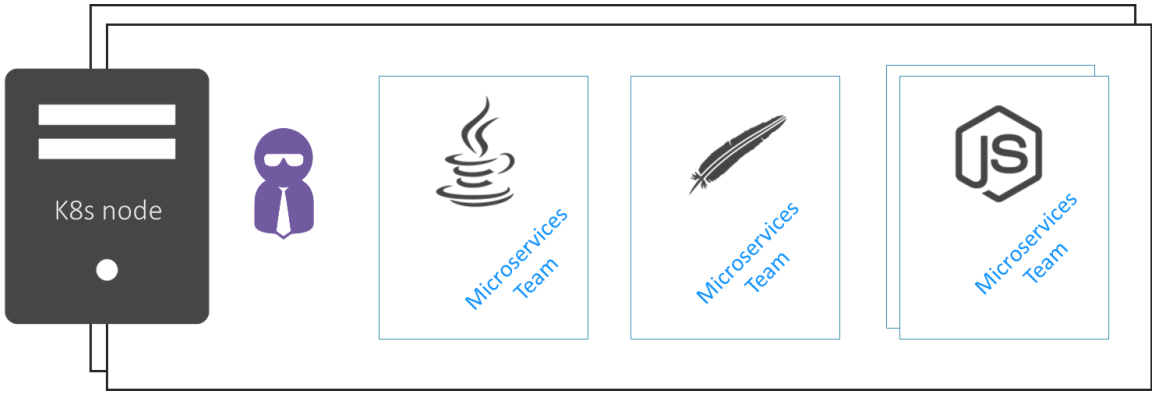
Questions

Do you have any questions about developing microservices 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



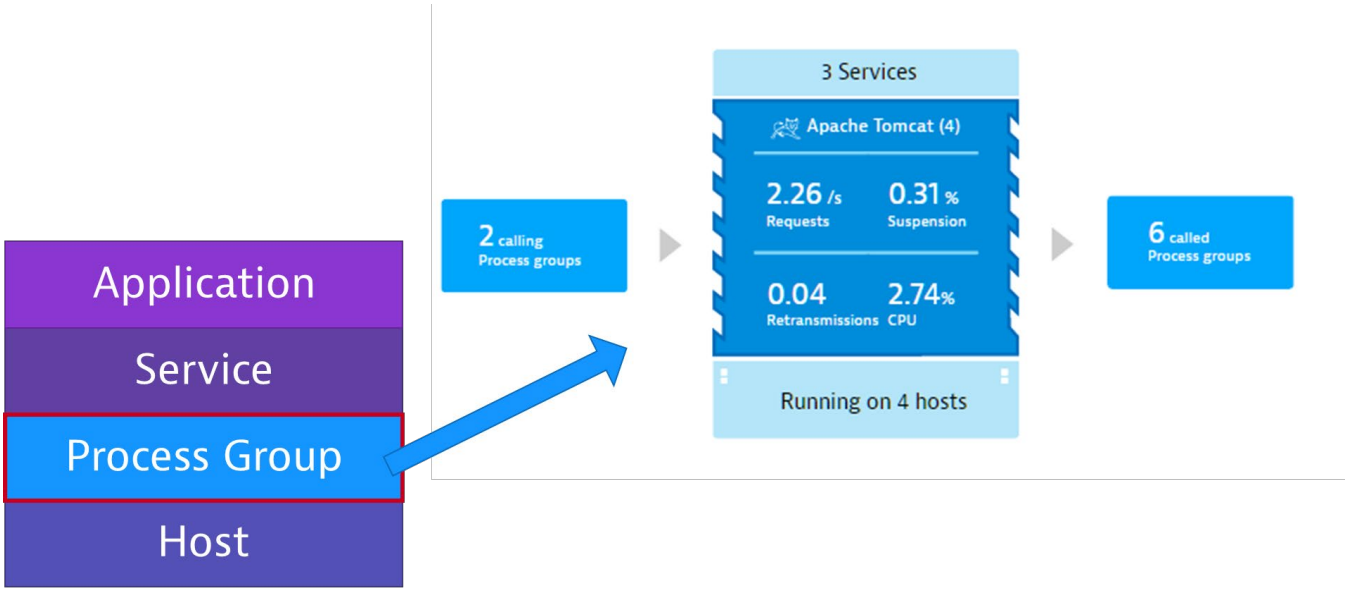
- Deploy one agent per host (kubectl create)
- Auto instrumentation of containerized microservices
- Auto distributed transaction tracing
- AI based root cause analytics

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

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?

Summary

- 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 _____.

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.

Introduction

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.

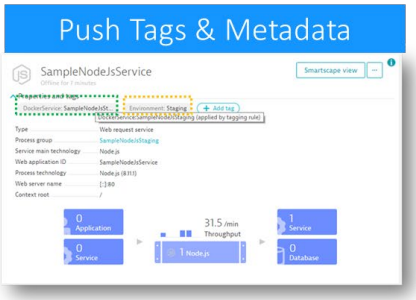
Use Cases – Key Takeaways

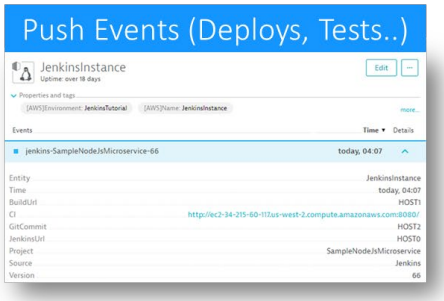
Push Testing Metrics to Dynatrace
Pass Test Script Context via Request Attributes
Automate Analysis
Compare Hotspots

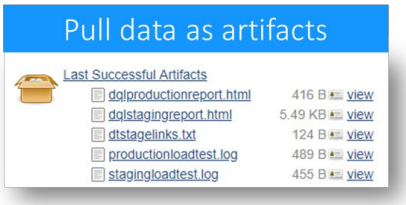
Load Testing Integration:

Taken from:
https://github.com/dynatrace-innovationlab/jenkins-dynatrace-pipeline-tutorial

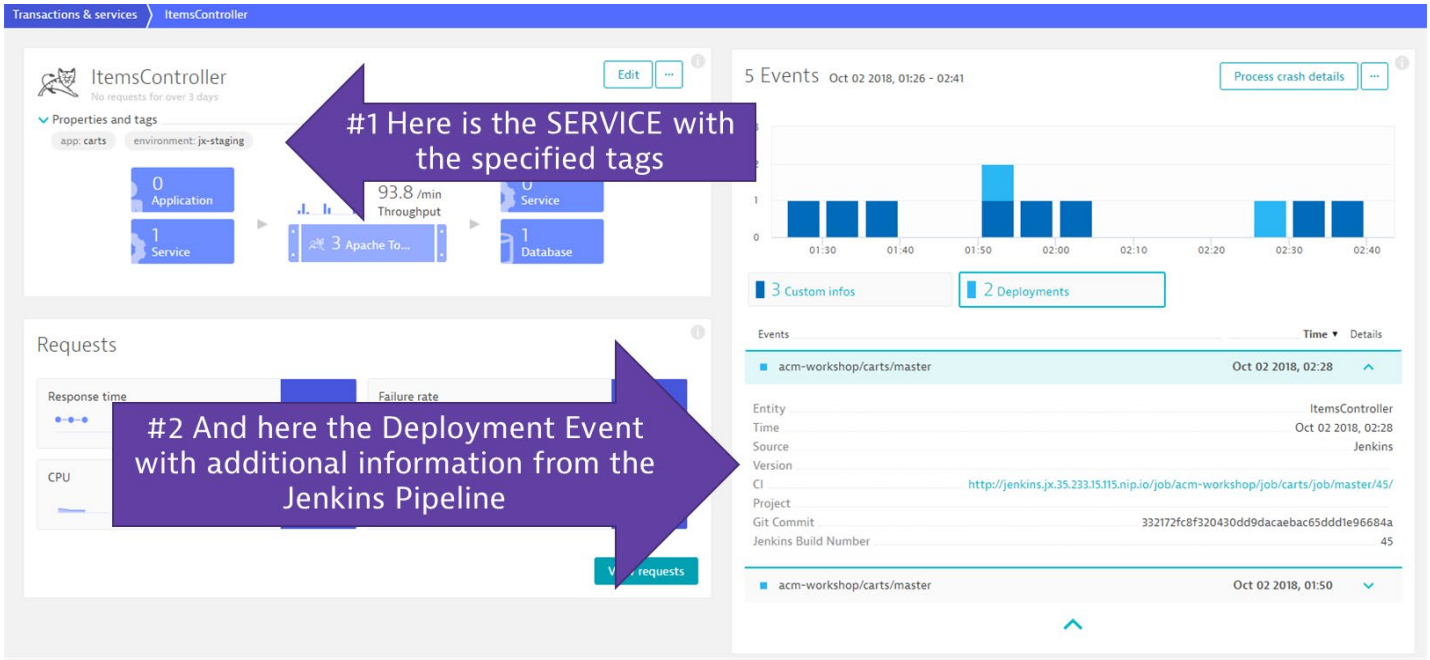
How does Jenkins and Dynatrace work together?







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-repository*07_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 requrest 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-repository*07_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:

Lab activity:

Performance as a Service Step 3 of 7

Define Performance Signature for Cart Service

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

Instructions

This lab is found in the GitHub workshop repository:

*course-repository*07_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 pipline 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

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

<table>
<tr><td colspan="2">Recreate
Shut down version A
Deploy version B after A is turned off</td></tr>
<tr><td>Pros:
• Easy to do
• Application state entirely renewed</td><td>Cons:
• Downtime – High impact on the user</td></tr>
<tr><td colspan="2">Ramped
aka Rolling upgrade
Replace instances with new version one by one</td></tr>
<tr><td>Pros:
• Easy to setup
• New version is slowly released
• Stateful applications can rebalance data</td><td>Cons:
• No control over traffic</td></tr>
<tr><td colspan="2">Blue/Green
Version B (green) is deployed alongside version A (blue)
Traffic is switched to green at load balancer
Blue is not immediately deleted</td></tr>
<tr><td>Pros:
• Instant rollout and rollback
• Avoiding version conflicts</td><td>Cons:
• Difficult for stateful applications
• Long running transactions</td></tr>
</table>

<table>
<tr><td colspan="2">Canary
Gradually shifting traffic from version A to B
E.g.: 90% A - 10% B for 10 mins, then 80% A - 20% B …
Controlled rollout with easy rollback
Criteria for traffic distribution</td></tr>
<tr><td>Pros:
• New version only released to a subset of users
• Tryouts under production conditions
• Fast and easy rollback</td><td>Cons:
• Difficult for stateful applications</td></tr>
<tr><td colspan="2">A/B Testing
No deployment strategy but related
Testing conversion rates of features
Criteria for traffic distribution</td></tr>
<tr><td>Pros:
• Run several versions in parallel
• Full traffic control</td><td>Cons:
• Requires intelligent load balancer (L7)
• Hard to troubleshoot</td></tr>
<tr><td colspan="2">Shadow
Deploy version B alongside version A
Mirror traffic to version B
Go live when quality requirements are met</td></tr>
<tr><td>Pros:
• Testing B under real conditions
• No user impacts
• Controlled rollout</td><td>Cons:
• Expensive – twice the resources
• Not a true user test
• May require mocking services</td></tr>
</table>

Istio architecture

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

Pilot

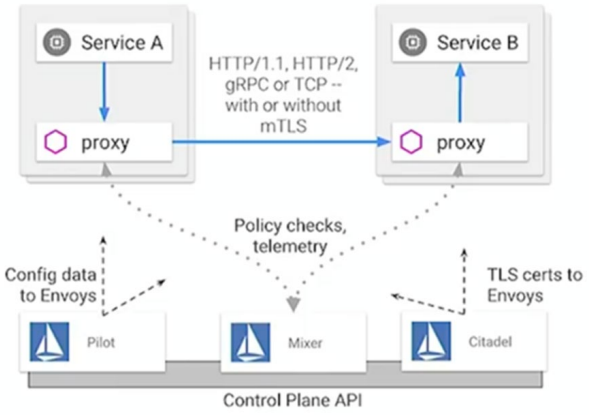
- Control plane to configure and push service communication policies
- Routing and forwarding policies

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

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: sockshop-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

VirtualService

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: sockshop
spec:
  hosts:
  - "*"
  gateways:
  - sockshop-gateway
  http:
  - route:
    - destination:
        host: front-end.production.svc.cluster.local
        subset: v1
```

dynatrace

DestinationRule

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: front-end-destination
spec:
  host: front-end.production.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

ServiceEntry

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: carts-and-orders-db
spec:
  hosts:
  - "*.mongodb.net"
  addresses:
  - 18.214.93.113
  - 18.215.19.51
  - 54.82.134.106
  ports:
  - number: 27017
    name: tcp
    protocol: TCP
  location: MESH_EXTERNAL
```

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 mess external traffic to leave the service mesh

Instructions

This lab is found in the GitHub workshop repository:

*course-repository*08_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-repositor*08_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:

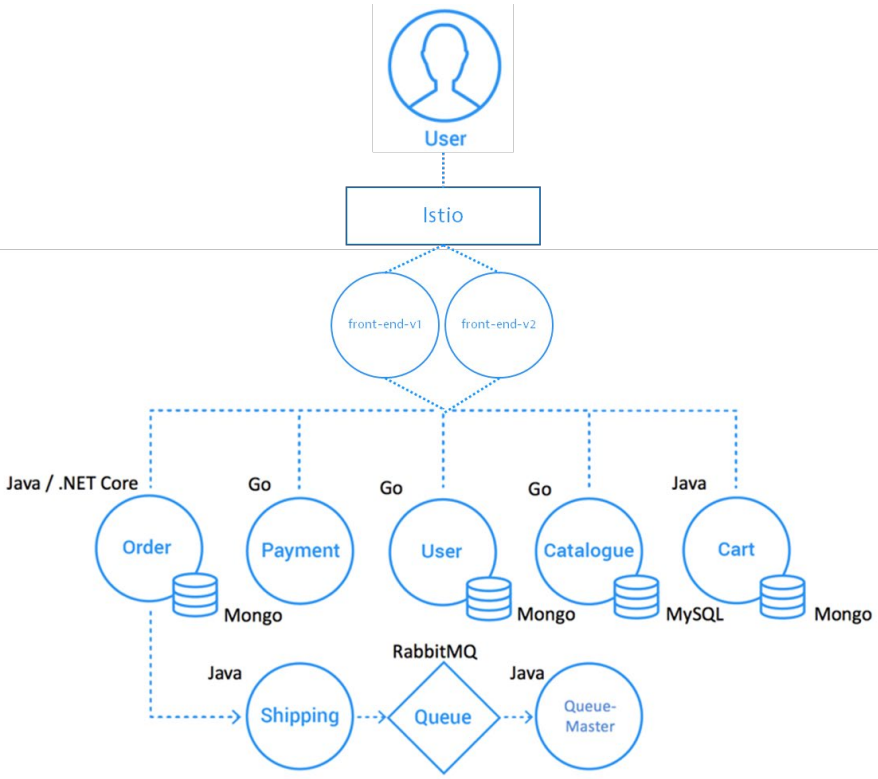
Lab activity:

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 wieght 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

Runbook Automation and Self-Healing

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

Runbooks:

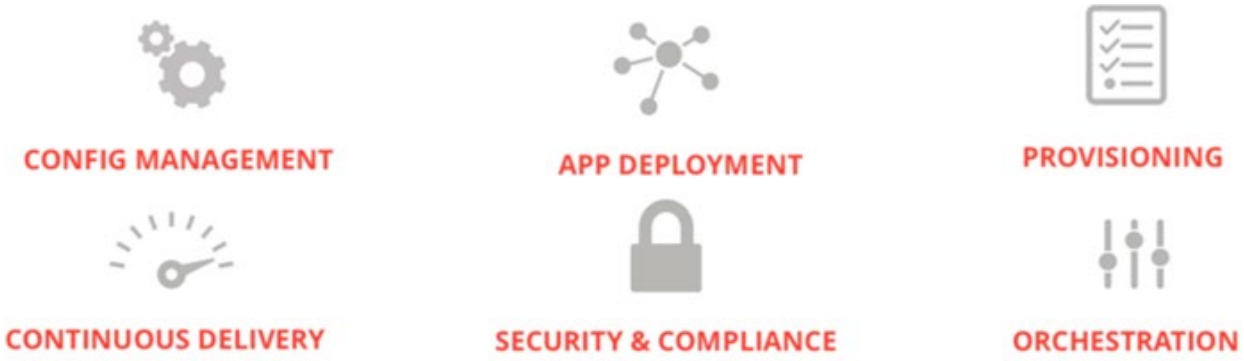
- "compilation of routine procedures and operations that the system operator (administrator) carries out"

Ansible:

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

Ansible Tower:

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



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-repository*09_Runbook_Automation_and_Self_Healing/04_Run_Playbook

Notes:

Summary

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

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.

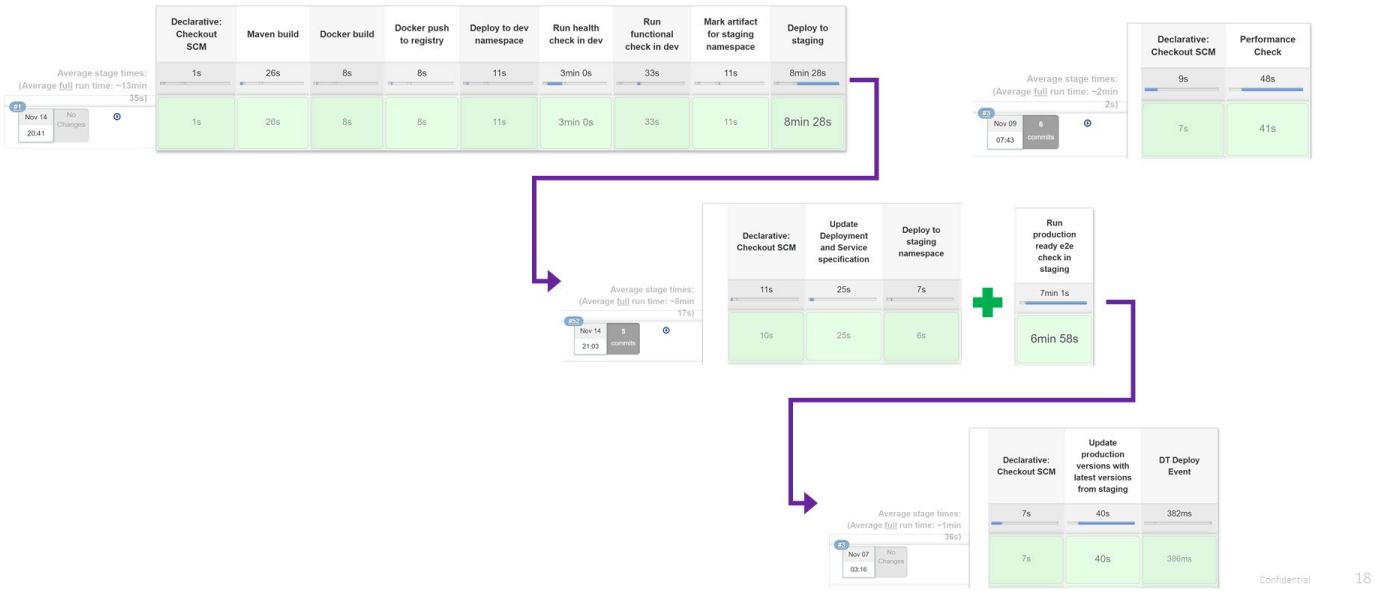
Shift-Right: Ability to push deployment information and meta data to your monitoring environment, e.g.: differentiate BLUE vs GREEN deployments, push build or revision number of deployment, notify about configuration changes.

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

Shift-Left: Break Pipeline Earlier



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:

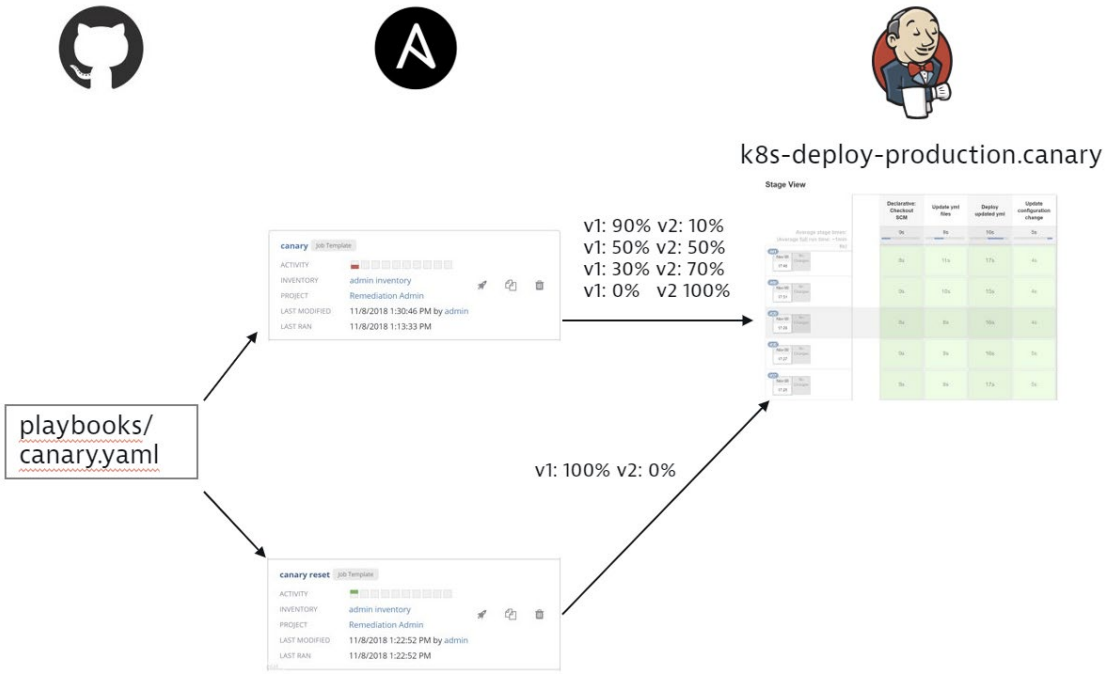
Unbreakable Delivery Pipeline

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:

1. Create Job Template for Canary Release in Ansible Tower
2. 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 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:

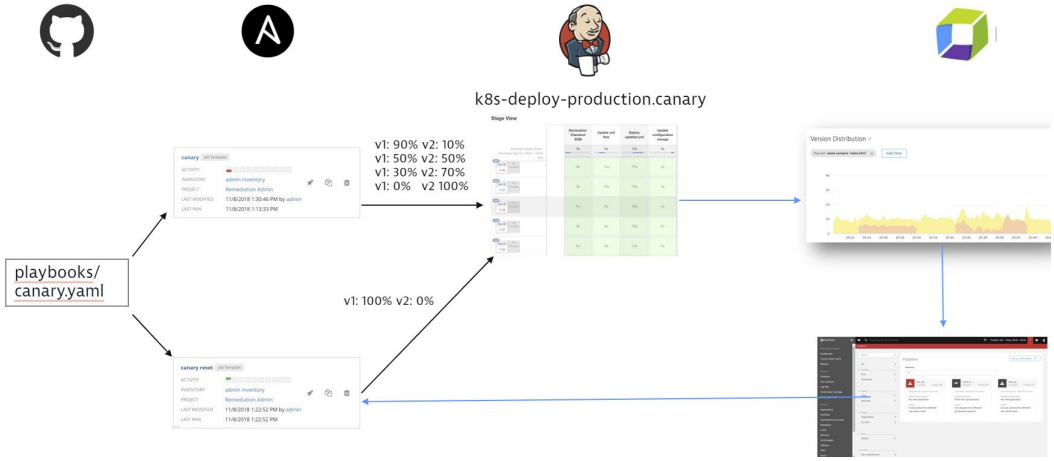
Unbreakable Delivery Pipeline

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 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 Probkem 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:

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

Questions

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

Lab activity:

Virtual Operations

Part 1 of 4

Check Prerequisites

In this lab lab you will explore the Dynatrace API. You will do this by perfomring 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

- _____ 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

You have reached the end of the planned exercises. Do you have any 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