

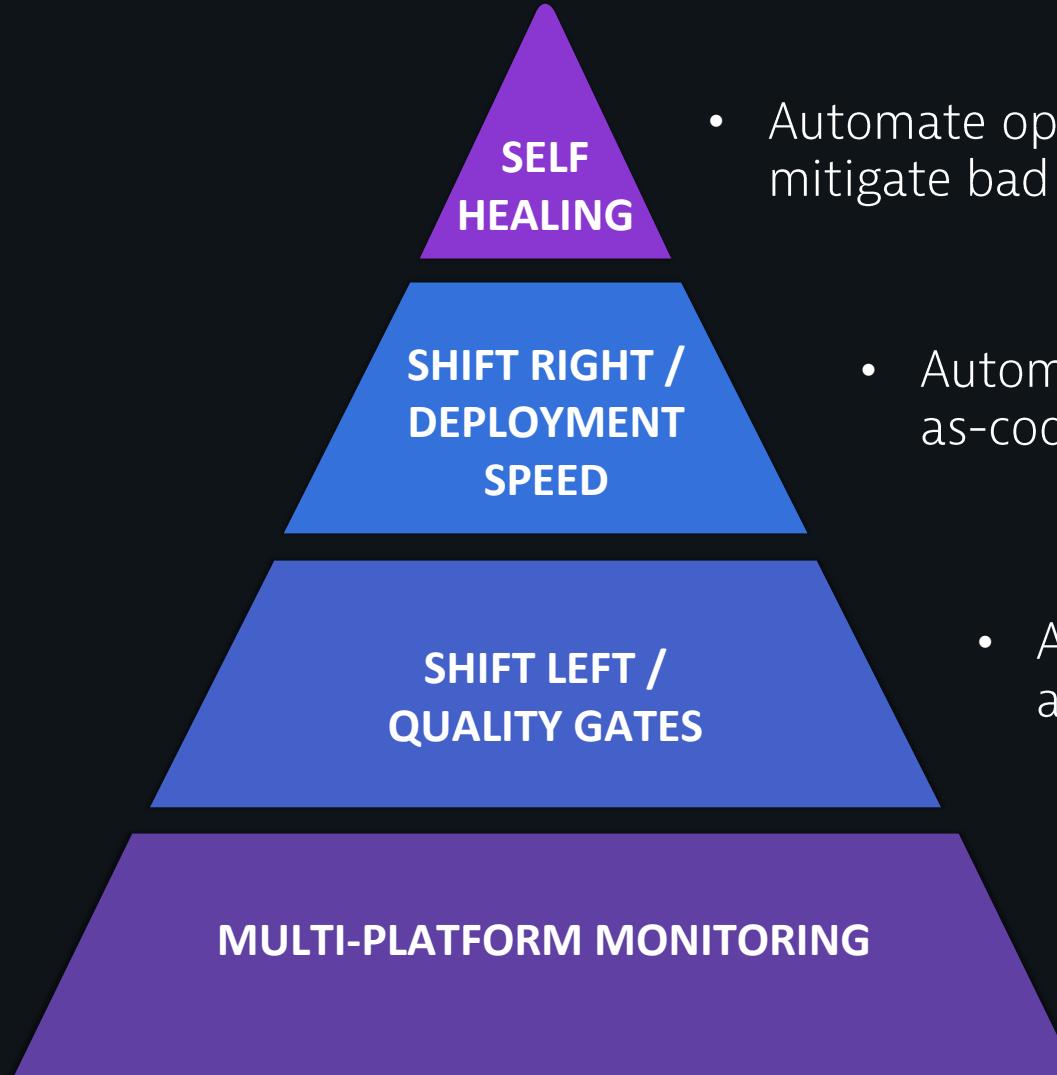
# Dynatrace Autonomous Cloud

---

GKE, Jenkins, Ansible Tower



# Key blue print – unbreakable continuous delivery model



- Automate operations (self-healing) – auto-mitigate bad deployments in production
- Automate deployment (shift-right) – push “monitoring-as-code” for auto-validation and auto-alerting
- Automate quality (shift-left) – automate the pipeline and stop bad code changes before they reach prod
- Automate monitoring – use monitoring strategically as feature of the end-to-end pipeline



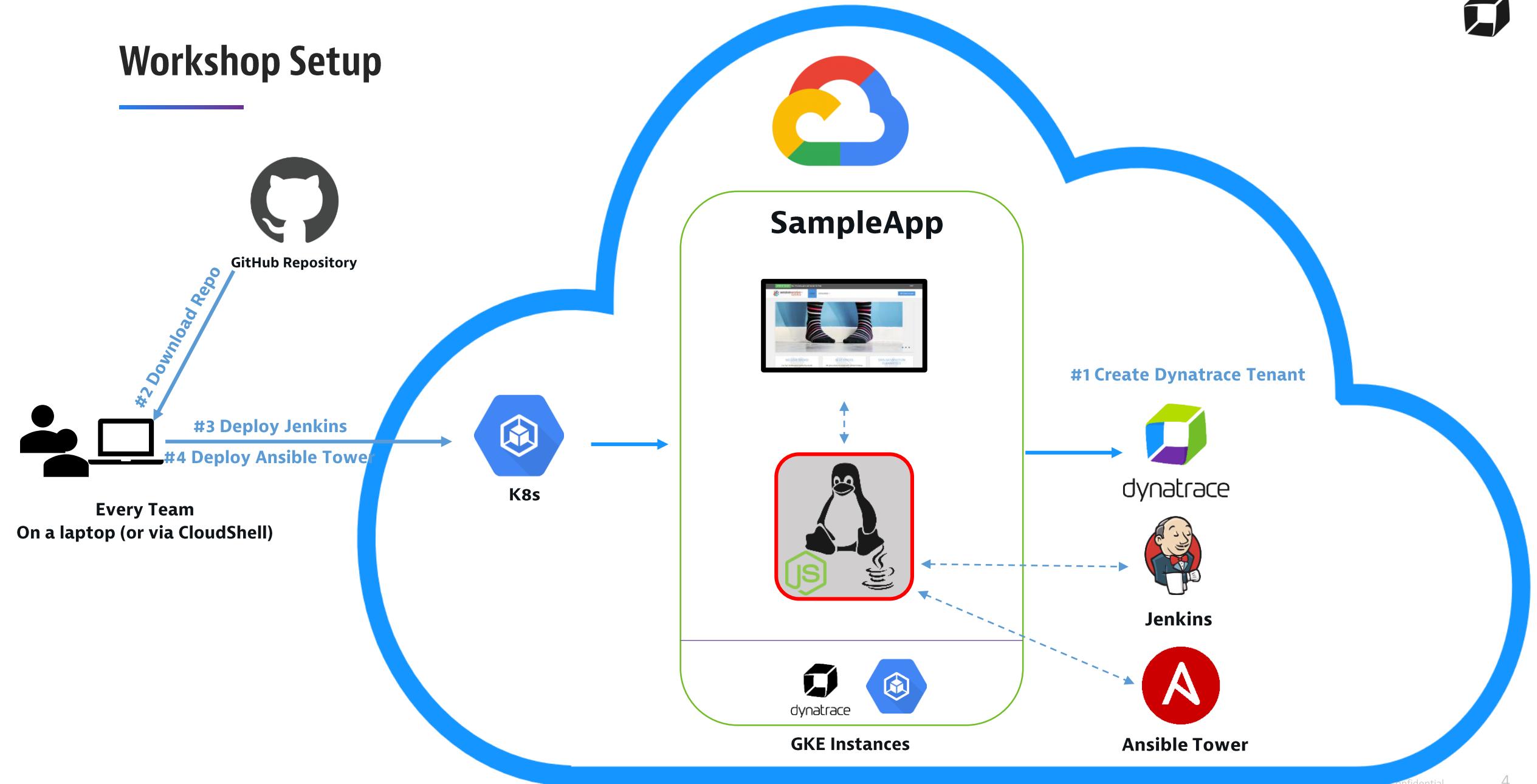
## What we will accomplish

---

- Deploying the SockShop application to our GKE cluster
- Setting up Jenkins on K8s in GKE
- Import a Jenkins Pipeline
- Setup Dynatrace Integration with Jenkins
  - Using Jenkins Plugin and Monspec
- Deploy different code builds and test their performance
  - Fail pipeline if code is not performant based on monspec
- Create a self-healing application using Dynatrace and Ansible Tower

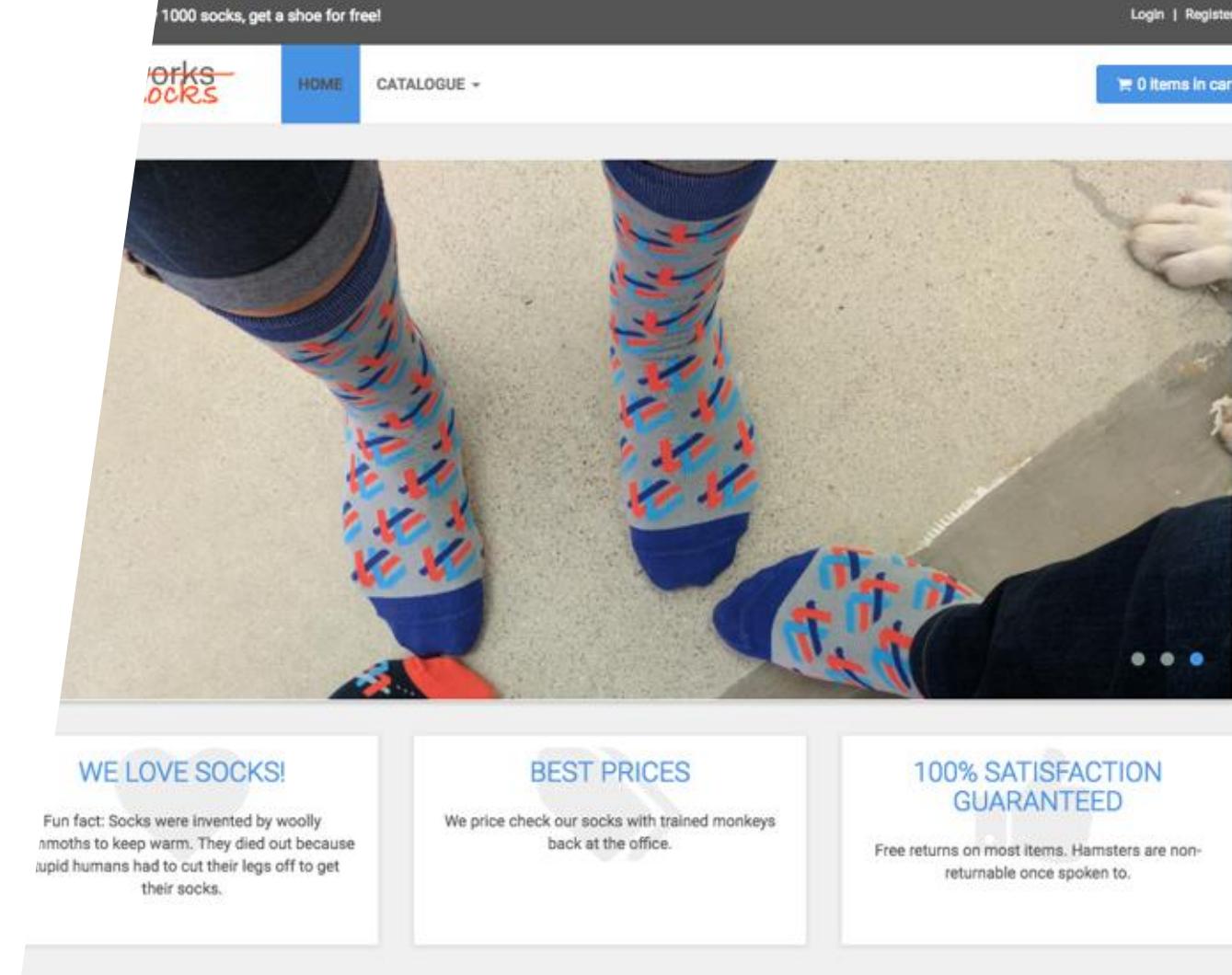


# Workshop Setup

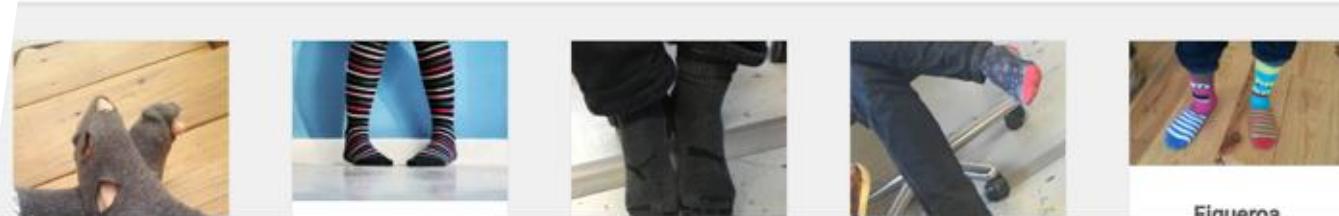


# Sock Shop Application

- Based on: <http://socks.weave.works>
- Opensource microservice sample app
- Polyglot
  - Java Spring Boot
  - Go
  - Node.js
  - RabbitMQ (order queuing)
  - MongoDB, MySQL (data stores)



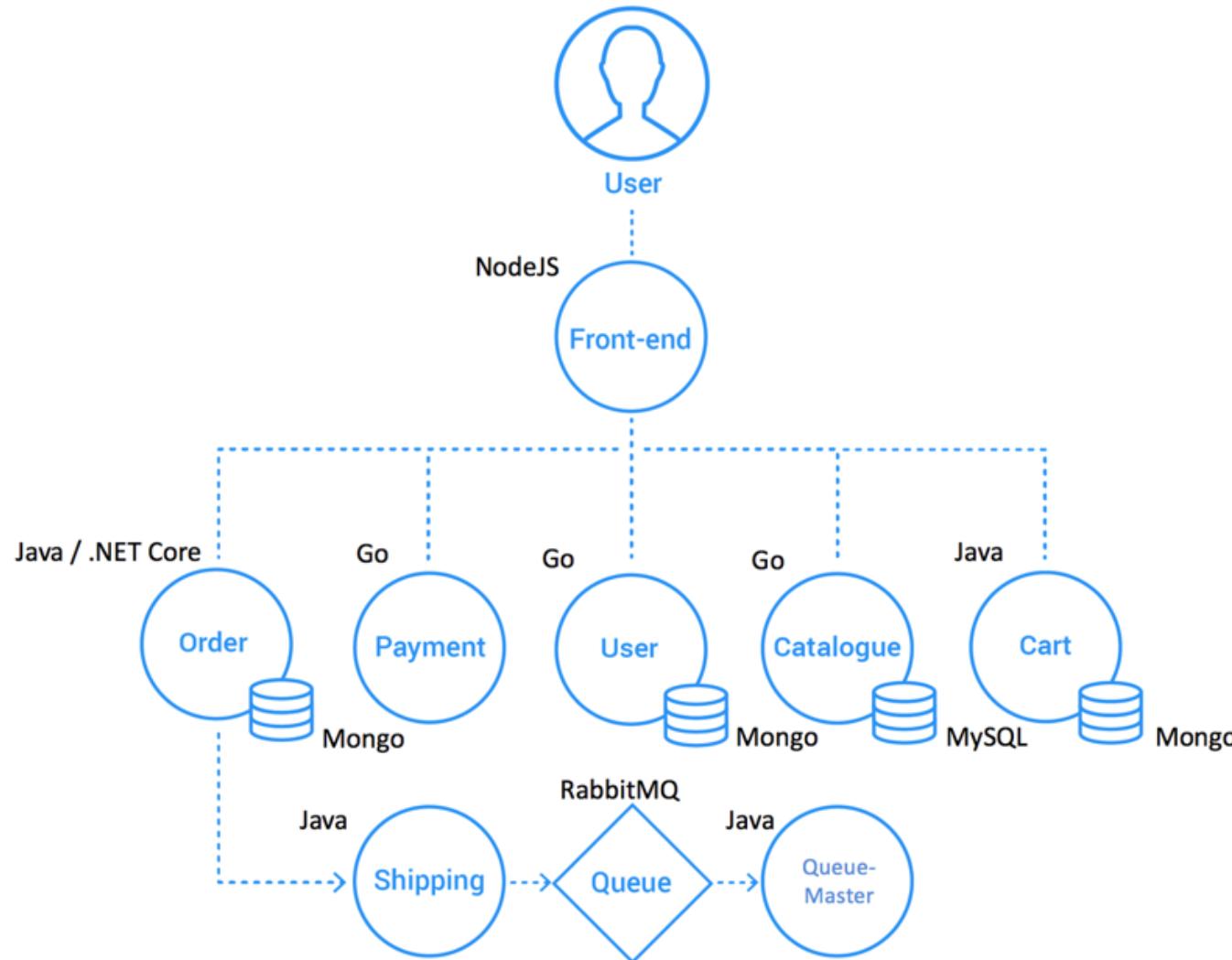
## HOT THIS WEEK





# Sock Shop architecture

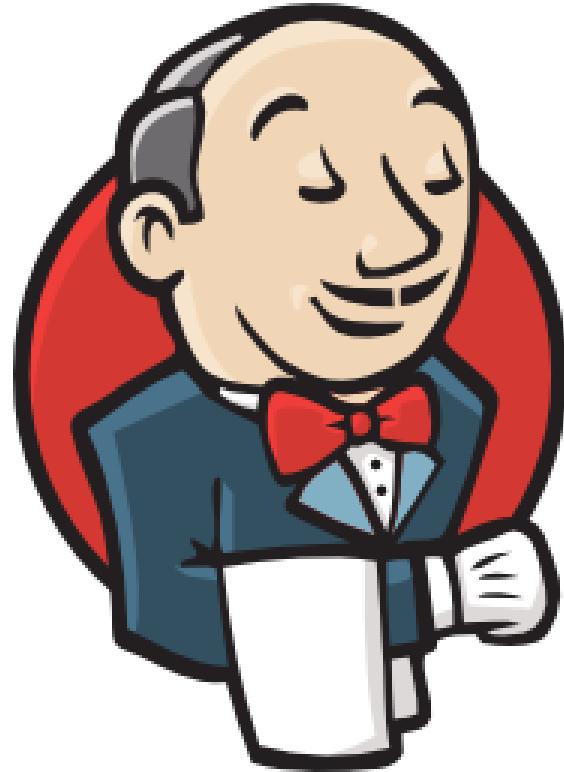
---





# Jenkins

---



- The Worlds most advanced BASH terminal
- In most cases it's fine to deploy to K8s via simple kubectl commands in k8s
  - Most of deployments probably done this way
  - Spoiler Alert: We're going to be doing this today
- No true "easy" way to determine a failure or if app is up and running

```
$kubectl wait -n dev deploy/myapp --for condition=available
```

This won't catch pods that didn't actually come online



# What our Pipeline will look like (hopefully)

## Stage View

|    | Checkout code | Build Code | Build Image | Push Image to Repo | Deploy to Staging | Run Load Test | Dynatrace Quality Gate |
|----|---------------|------------|-------------|--------------------|-------------------|---------------|------------------------|
|    |               |            |             |                    |                   |               | 1s                     |
| #7 | 679ms         | 6s         | 3s          | 3s                 | 5min 58s          | 5min 21s      | 1s<br>failed           |
| #6 | 1s            | 7s         | 3s          | 3s                 | 5min 58s          | 3min 47s      | 1s                     |

Average stage times:  
(Average full run time: ~11min 29s)

#7 Jul 19 11:28 No Changes

#6 Jul 19 11:16 1 commit

# Let's Start the fun

---



# Setup our GKE Cluster

- Login to your GCP Account

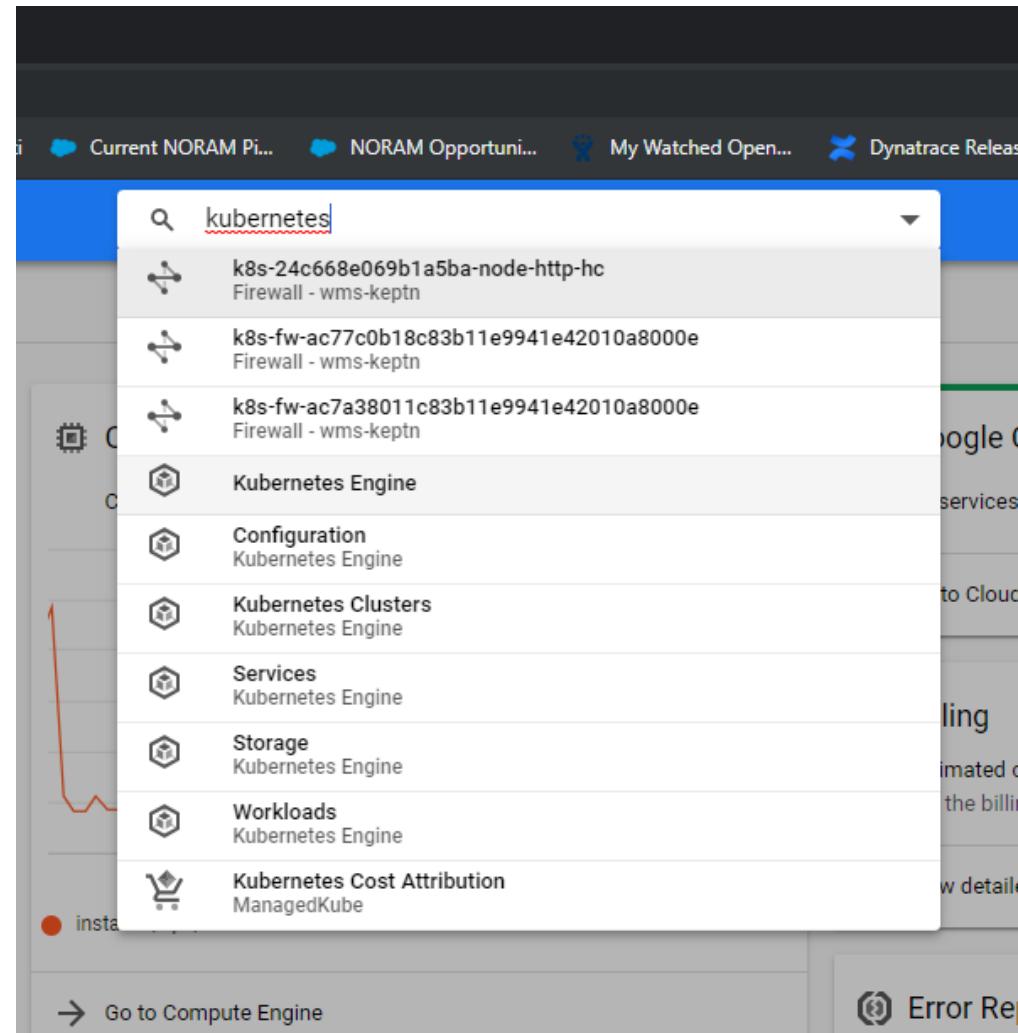
The screenshot shows the Google Cloud Platform dashboard for the project 'wms-keptn'. The dashboard is divided into several sections:

- Project info:** Shows the project name (wms-keptn), Project ID (wms-keptn), and Project number (963637366096). It also has a link to 'ADD PEOPLE TO THIS PROJECT' and 'Go to project settings'.
- Compute Engine:** A chart showing CPU utilization over time. The Y-axis ranges from 0 to 1.2, and the X-axis shows times from 4 PM to 4:45. A red line indicates utilization, with a note: 'instance/cpu/utilization: 0.168'. Below the chart is a link to 'Go to Compute Engine'.
- Google Cloud Platform status:** Shows 'All services normal' and a link to 'Go to Cloud status dashboard'.
- Billing:** Shows estimated charges for the billing period Aug 1 – 26, 2019, totaling USD \$0.00. There is a link to 'View detailed charges'.
- Error Reporting:** Shows 'No application errors in the last 24 hours' and a link to 'Go to Error Reporting'.
- APIs:** A chart showing API requests over time. The Y-axis ranges from 0 to 1.4, and the X-axis shows times from 4 PM to 4:45. A blue line indicates requests, with a note: 'Requests: 0.593'. Below the chart is a link to 'Go to APIs overview'.
- Trace:** Shows 'No trace data from the past 7 days' and a link to 'Get started with Stackdriver Trace'.
- Getting Started:** Links to 'Explore and enable APIs', 'Deploy a prebuilt solution', 'Add dynamic logging to a running application', and 'Monitor errors with Error Reporting'.
- News:** A section with news items:
  - Data engineering lessons from Google AdSense: using streaming joins in a recommendation system (4 hours ago)
  - Making the modern OS accessible for every enterprise (8 hours ago)
  - Skip the maintenance, speed up queries with BigQuery's clustering (3 days ago)A link 'Read all news' is at the bottom.



# Setup our GKE Cluster

- In the search box, type “Kubernetes” and select “Kubernetes Engine” from the search results
- This may take a few minutes if this is the first time you are attempting to launch Kubernetes in this account.

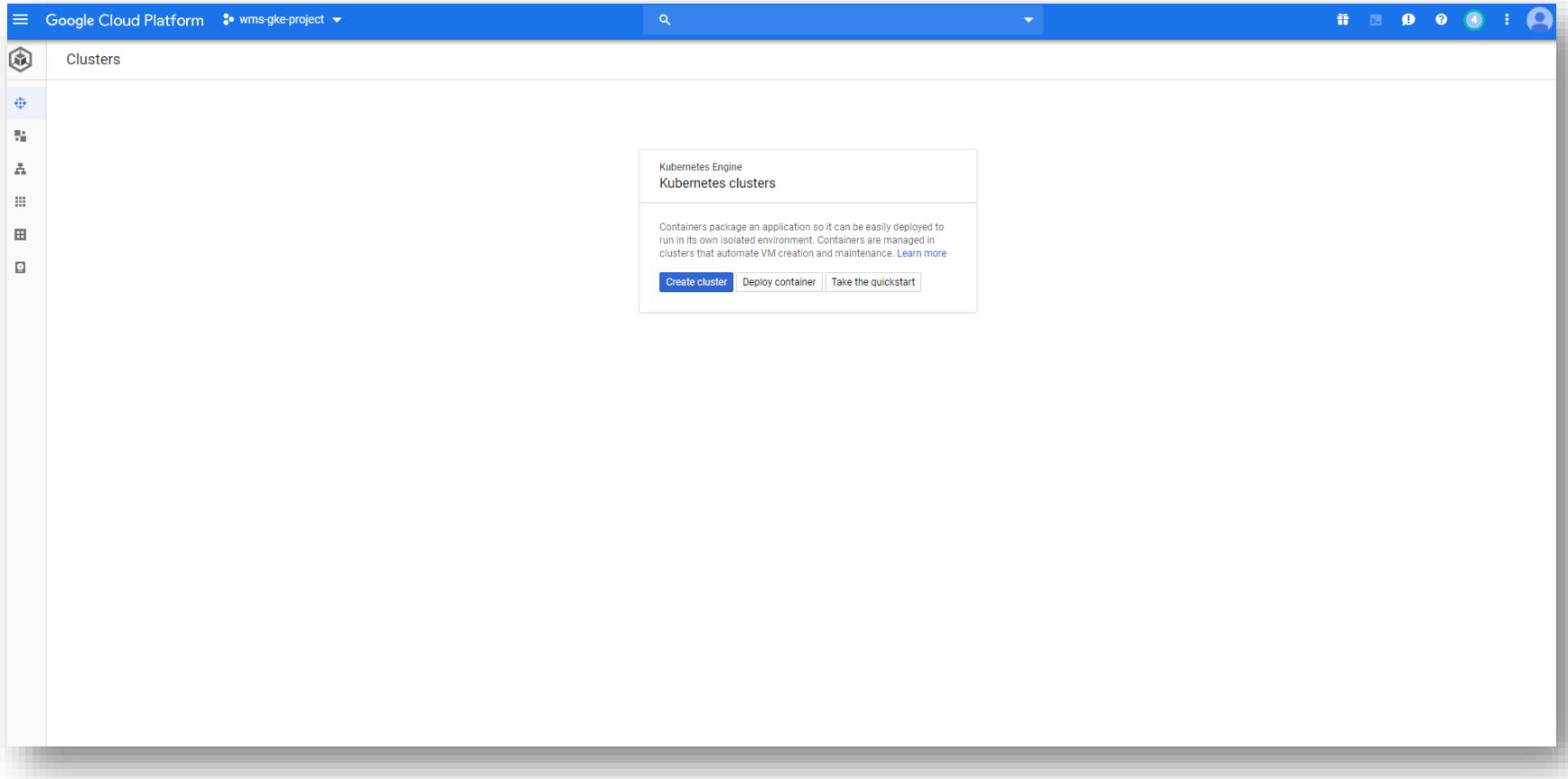




# Setup our GKE Cluster

---

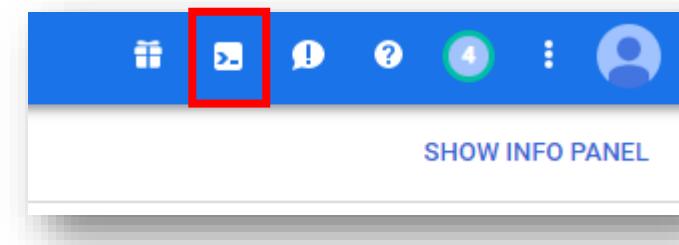
- After the initial APIs are setup, your Kubernetes Engine screen should look like this (unless you already have an existing cluster setup)





# Setup our GKE Cluster – Launching Cloudshell

- To launch and interact with our Kubernetes cluster we use Cloudshell
  - Cloudshell provides command-line access to your cloud resources directly from your browser
  - No need for a separate bastion host
  - Required tools pre-installed (git, docker, kubectl, etc...)
  - Persistent storage



```
Welcome to Cloud Shell! Type "help" to get started.  
Your Cloud Platform project in this session is set to wms-keptn.  
Use "gcloud config set project [PROJECT_ID]" to change to a different project.  
wmsgclouddemouser1@cloudshell:~ (wms-keptn)$
```



# Setup our GKE Cluster – Cloning our repository

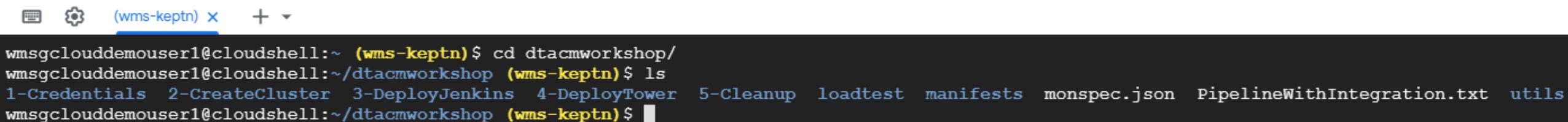
---

- In Cloudshell, clone/download the following GitHub Repository:

<https://github.com/Dynatrace-APAC/workshop-dtac>

Run the command: `git clone https://github.com/Dynatrace-APAC/workshop-dtac.git`

- This will create a folder called **workshop-dtac**
- This will contain the scripts we will use for the remainder of the workshop



```
wmsgclouddemouser1@cloudshell:~ (wms-keptn)$ cd dtacmworkshop/
wmsgclouddemouser1@cloudshell:~/dtacmworkshop (wms-keptn)$ ls
1-Credentials 2>CreateCluster 3-DeployJenkins 4-DeployTower 5-Cleanup loadtest manifests monspec.json PipelineWithIntegration.txt utils
wmsgclouddemouser1@cloudshell:~/dtacmworkshop (wms-keptn)$ █
```



# Dynatrace Credentials – Tenant ID & Environment ID

---

- In order to proceed further we need to make note of our Dynatrace Tenant/Environment and API/PaaS Tokens
- For the Tenant ID, you can find it in the first part of your URL
  - [https://<TENANT\\_ID>.live.dynatrace.com](https://<TENANT_ID>.live.dynatrace.com) – SaaS Deployments
    - For example, for <https://jwx05250.live.dynatrace.com>, Tenant ID=jwx05250
  - [https://<TENANT\\_ID>.dynatrace-managed.com](https://<TENANT_ID>.dynatrace-managed.com)
    - For example, for <https://abc123.Dynatrace-managed.com>, Tenant ID=abc123
- The Environment ID is only applicable for Dynatrace Managed Deployments. You can find this value in the second half of your URL
  - [https://<TENANT\\_ID>.dynatrace-managed.com/e/<ENVIRONMENT\\_ID>](https://<TENANT_ID>.dynatrace-managed.com/e/<ENVIRONMENT_ID>)
    - For example, for <https://abc1234.Dynatrace-managed.com/e/1234-5678>, Environment ID=1234-5678



# Dynatrace Credentials - API Token

- Go in Settings -> Integration -> Dynatrace API
  1. Click on Generate Token
  2. Enter a name for your token (e.g. GKE)
  3. Enable all the access scope
  4. Don't forget to click on the Save button

The screenshot shows the 'API tokens' section of the Dynatrace interface. A new token is being generated:

1. The 'Generate token' button is highlighted with a red box.
2. The 'Token name' field contains 'OpenShiftOperator' and is highlighted with a red box.
3. The 'Generated token' field contains '1kjTmDPERQ2RnsOsdUxYV' and has a 'Copy' button next to it, both highlighted with a red box.
4. The 'Save' button at the bottom right is highlighted with a red box.

The page also includes a table header with columns for 'API tokens', 'Owner', and actions like 'Disable/enable', 'Delete', and 'Edit'. The owner listed is 'student0898@trial.dynatracelabs.com'.



# Dynatrace Credentials - PaaS Token

- Go in Settings -> Integration -> Platform as a Service
  1. Either copy the existing InstallerDownload token or click on Generate Token
  2. Enter a name for your token (e.g. GKE), click Save

The screenshot shows the 'My Dynatrace PaaS tokens' section. It includes tabs for 'My Dynatrace PaaS tokens' and 'Other Dynatrace PaaS tokens'. Below the tabs is a header for 'Platform as a Service tokens'.

| Token name            | Owner                               | Disable/enable                      | Delete | Edit |
|-----------------------|-------------------------------------|-------------------------------------|--------|------|
| InstallerDownload     | student0898@trial.dynatracelabs.com | <input checked="" type="checkbox"/> | X      | ▼    |
| OpenshiftOperatorPaaS | student0898@trial.dynatracelabs.com | <input checked="" type="checkbox"/> | X      | ▲    |

A modal window titled 'Generated token' is open, displaying the token value '6v4gzGwwRp6XgsJvMcISx' in a text input field. A red box labeled '1' highlights the 'Generate token' button. Another red box labeled '2' highlights the 'Token name' input field containing 'OpenshiftOperatorPaaS'. A third red box labeled '3' highlights the 'Copy' button next to the token value.



# Creating our Credentials

---

- We now need to store a local copy of our Dynatrace Tenant and API info for use when creating our cluster
- Navigate to the directory **1-Credentials** and execute the script **defineCredentials.sh**
  - This will ask for your Dynatrace Tenant/Environment information as well as your API and PaaS tokens
    - These values can be obtained from your Dynatrace tenant (see previous slides)
  - Once you have entered the values and confirmed they are correct, we can move on to creating our cluster

```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop (wms-keptn)$ cd 1-Credentials/
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/1-Credentials (wms-keptn)$ ls
creds.json creds.sav defineCredentials.sh
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/1-Credentials (wms-keptn)$ ./defineCredentials.sh
Please enter the credentials as requested below:
Dynatrace Tenant ID (ex. https://<TENANT_ID>.live.dynatrace.com or https://<TENANT_ID>.dynatrace-managed.com): abc123
Dynatrace Environment ID (Dynatrace Managed Only - https://<TENANT_ID>.dynatrace-managed.com/e/<ENVIRONMENT_ID>): 1234567
Dynatrace API Token: 123456789
Dynatrace PaaS Token: 9876544
```



# Setup our GKE Cluster

---

- Back in your GCP account, launch a Cloudshell session
- Navigate to the directory **2-CreateCluster**
- Execute the script **setupenv.sh** and confirm that your credentials are correct

```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop (wms-keptn)$ cd 2-CreateCluster/  
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2-CreateCluster (wms-keptn)$ ls  
setupenv.sh  
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2-CreateCluster (wms-keptn)$ ./setupenv.sh
```

This step will take about 10 minutes to complete



# Setup our GKE Cluster

---

- The **setupenv.sh** script will do the following:
  - Create a new GKE cluster
  - Deploy the Dynatrace OneAgent Operator to the cluster
  - Deploy the SockShop application to our cluster
    - Including creating the required Namespaces, storage objects and LoadBalancers



# Validate our Cluster setup

- After about 10 minutes since running the `setupenv.sh` script, let's verify that everything is up and running
- Let's check and make sure our Deployments are Ready, run the following command in Cloudshell:
  - `kubectl get deployments -n dev`
- **Note:** This can also be validated in the GKE Console by selecting “Workloads” from the menu on the left side of the screen

```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2-CreateCluster (wms-keptn)$ kubectl get deployments -n dev
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
carts        1          1          1           1          90m
carts-db     1          1          1           1          90m
catalogue    1          1          1           1          90m
catalogue-db 1          1          1           1          90m
front-end    1          1          1           1          90m
orders       1          1          1           1          90m
orders-db    1          1          1           1          90m
payment      1          1          1           1          90m
queue-master 1          1          1           1          90m
rabbitmq     1          1          1           1          90m
shipping     1          1          1           1          90m
user         1          1          1           1          90m
user-db      1          1          1           1          90m
```



# Validate our Cluster setup

---

- Lets ensure that the SockShop application services are also created by running the following command:
  - `$kubectl get svc -n dev -o wide`
- Our Carts and Front-end services should have External-IP addresses assigned
- **Note:** This can also be validated in the GKE Console by selecting “Services & Ingress” from the menu on the left side of the screen

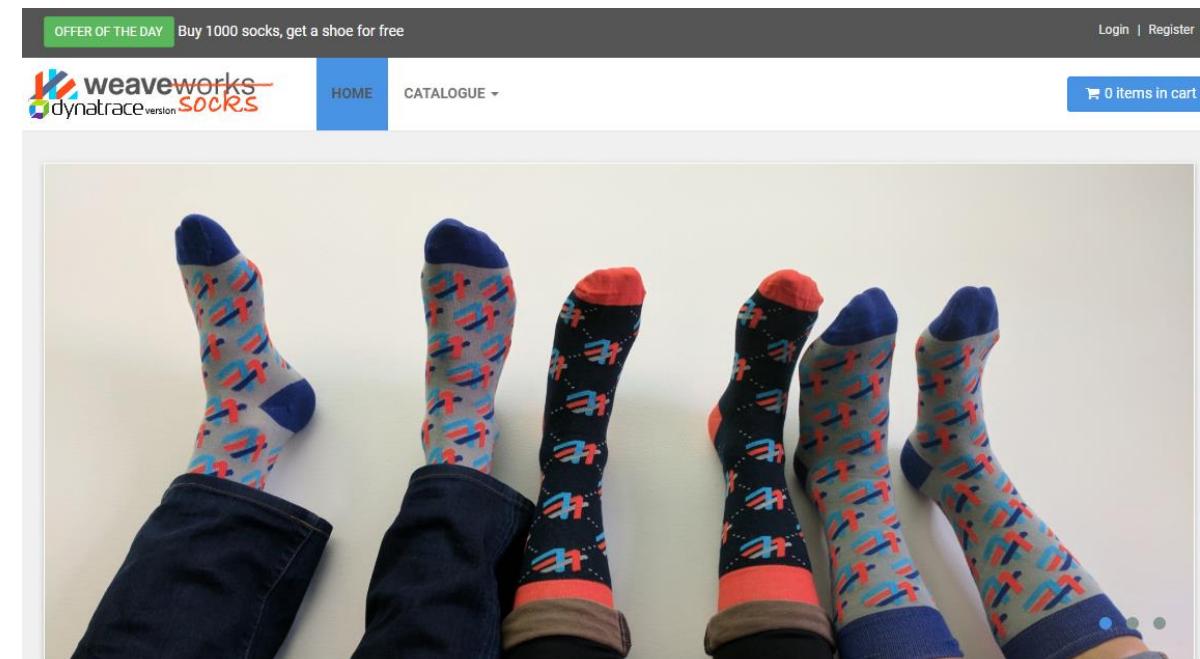
```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2>CreateCluster (wms-keptn)$ kubectl get svc -n dev -o wide
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)           AGE   SELECTOR
carts         LoadBalancer 10.31.250.178  35.232.184.170  80:30515/TCP   92m   app=carts
carts-db      ClusterIP   10.31.240.167  <none>        27017/TCP      92m   name=carts-db
catalogue     ClusterIP   10.31.242.33   <none>        80/TCP          92m   app=catalogue
catalogue-db  ClusterIP   10.31.248.14   <none>        3306/TCP       92m   name=catalogue-db
front-end     LoadBalancer 10.31.252.209  104.197.12.209  8080:30981/TCP  92m   app=front-end
orders        ClusterIP   10.31.255.165  <none>        80/TCP          92m   app=orders
orders-db     ClusterIP   10.31.255.198  <none>        27017/TCP      92m   name=orders-db
payment       ClusterIP   10.31.247.35   <none>        80/TCP          92m   app=payment
queue-master  ClusterIP   10.31.249.206  <none>        80/TCP          92m   app=queue-master
rabbitmq      ClusterIP   10.31.241.221  <none>        5672/TCP,9090/TCP  92m   name=rabbitmq
shipping      ClusterIP   10.31.246.77   <none>        80/TCP          92m   app=shipping
user          ClusterIP   10.31.252.24   <none>        80/TCP          92m   app=user
user-db       ClusterIP   10.31.241.83   <none>        27017/TCP      92m   io.kompose.service=user-db
```



# Validate our Cluster setup

---

- Let's launch our application in a web browser by copying and pasting the External-IP and Port of the front-end service. In the previous example it would be: <http://104.197.12.209:8080>
- The SockShop homepage should appear in your browser



The screenshot shows the homepage of the SockShop website. At the top, there is a navigation bar with links for 'OFFER OF THE DAY' (Buy 1000 socks, get a shoe for free), 'HOME', 'CATALOGUE', and a shopping cart icon showing '0 items in cart'. The main content area features a grid of socks. Below the grid, there are three call-to-action boxes: 'WE LOVE SOCKS!' with a fun fact about their invention, 'BEST PRICES' with a note about price checking, and '100% SATISFACTION GUARANTEED' with a note about returns.

OFFER OF THE DAY Buy 1000 socks, get a shoe for free

weaveworks  
dynatrace socks

HOME CATALOGUE

0 items in cart

WE LOVE SOCKS!

Fun fact: Socks were invented by woolly mammoths to keep warm. They died out because stupid humans had to cut their legs off to get their socks.

BEST PRICES

We price check our socks with trained monkeys back at the office.

100% SATISFACTION GUARANTEED

Free returns on most items. Hamsters are non-returnable once spoken to.



## Clean up script placeholder

---

- In Terminal/Cloudshell execute the script: **cleanup.sh**
- This will remove the dev, production and istio-system namespaces (and their associated objects) so we have a clean cluster
- After this completes, we will redeploy the sock shop application in the dev namespace. We only need to execute the script: **deploy-sockshop.sh** and then wait a few minutes
- Then check if the SockShop app is back up and running

```
$kubectl get svc -n dev -o wide
```



# Validate Dynatrace

- Now that we know our application is deployed and running successfully, we can validate that everything is being displayed in Dynatrace
- Navigate back to Dynatrace and you should see data for Host, Process and Services

The Dynatrace Home screen displays various monitoring metrics:

- Quick overview:** Shows 0 Problems, 1 Application health issue (All fine), 0 Synthetic monitor issues, 2 Database health issues (All fine), and 39 Processes.
- Application health:** Worldmap (Apdex) showing "My web application" as the most active application. A note says "Only private / Internal IP addresses have been detected. Map these to real locations." with a magnifying glass icon.
- Infrastructure:** Host health shows 1 All fine host. Network status shows 10 Processes (Talkers) and 1 Host, with a volume of 252 kbit/s. AWS section shows 1/1 RDS instances, 1/12 Load balancers, and 2/15 EC2 instances. Docker section shows 1 Docker host with 59 containers and 24 images.
- Smartscape:** A hexagonal grid visualization showing 10 All fine hosts.

The Dynatrace Transactions & services screen shows a list of 10 Services:

| Name                  | Type      | Status                 |
|-----------------------|-----------|------------------------|
| :8080 catalogue       | catalogue | *                      |
| :8080 user            | user      | *                      |
| :8080 payment         | payment   | *                      |
| carts                 | app.jar   | carts*                 |
| front-end             | server.js | (front-end) front-end* |
| HealthCheckController | app.jar   | orders*                |
| HealthCheckController | app.jar   | queue-master*          |
| ItemsController       | app.jar   | carts*                 |

# Deploying Jenkins

---



# Deploy Jenkins

---

- In Terminal/Cloudshell execute the script: **deployJenkins.sh** in the directory **3-DeployJenkins**
- This will deploy and startup Jenkins, import our pipeline and output the credentials

```
wmsgclouddemouser1@cloudshell:~/dynatrace-sockshop (wms-keptn) $ ./deployJenkins.sh
namespace/cicd created
persistentvolumeclaim/jenkins-jobs created
persistentvolumeclaim/jenkins-workspaces created
persistentvolumeclaim/maven-repo created
service/jenkins created
deployment.extensions/jenkins-deployment created
clusterrolebinding.rbac.authorization.k8s.io/jenkins-rbac created
secret/jenkins-secret created
Waiting for Jenkins to start...
-----
Jenkins is running at : http://35.226.212.58:24711
Username is : admin
Password is : AiTx4u8VyUV8tCKk
```

**Note: You should change the default password!**



# Jenkins Setup

- After Jenkins has successfully deployed, navigate to the Jenkins URL and login with the previously provided username and password (again, recommend you change the password 😊)

Screenshot of the Jenkins dashboard:

The dashboard shows the following information:

- Top Bar:** Jenkins logo, search bar (6 results), admin log out.
- Left Sidebar:** New Item, People, Build History, Manage Jenkins, My Views, Lockable Resources, Credentials, New View.
- Build Queue:** No builds in the queue.
- Build Executor Status:** 1 Idle, 2 Idle.
- Central View:** A table showing build status for DeploySockShop. It includes columns: S (Status), W (Icon), Name (DeploySockShop), Last Success (N/A), Last Failure (N/A), and Last Duration (N/A). There is also a "add description" link and a "Legend" section with RSS links for all, failures, and latest builds.



# Run our first pipeline

---

- On the Jenkins main screen we should see the "DeploySockShop" pipeline, select the pipeline by clicking it

The screenshot shows the Jenkins interface for the "Pipeline DeploySockShop". The left sidebar contains navigation links: Back to Dashboard, Status, Changes, Build with Parameters, Delete Pipeline, Configure, Full Stage View, Rename, Performance Signature, and Pipeline Syntax. The main content area is titled "Pipeline DeploySockShop" and displays the message "This build requires parameters:". Below this is a dropdown menu labeled "BUILD One ▾" with the option "Choose Build" underneath. A large blue button labeled "Build" is highlighted with a blue arrow pointing to it from the bottom right, and the text "Select Build ‘One’ and Build" is overlaid on the arrow.

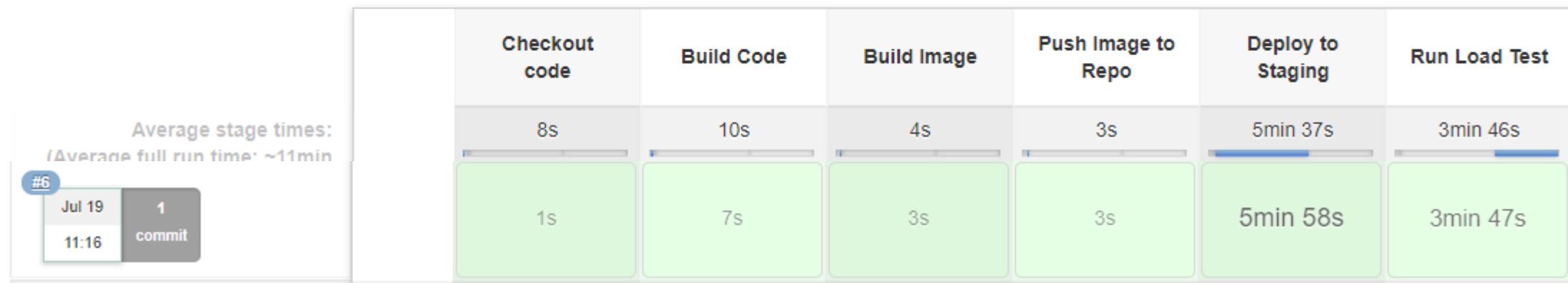


## If all goes well...

---

- The pipeline should run successfully (Note: this will take about 10 min)

### Stage View



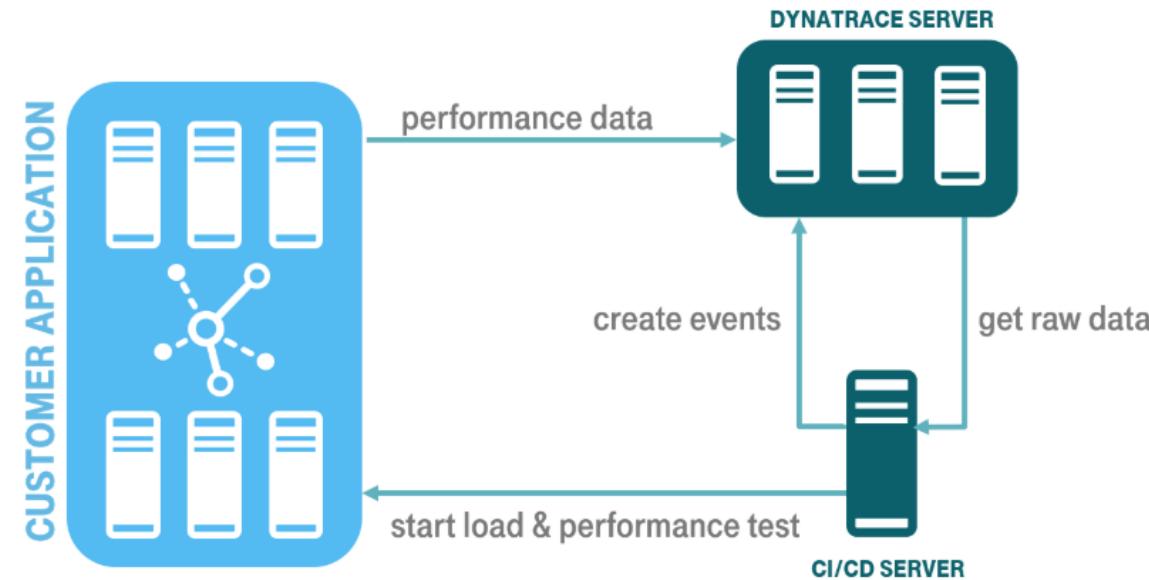
# Dynatrace Integration

---



# Dynatrace Integration

- GitHub Repo and Example: <https://github.com/jenkinsci/performance-signature-dynatrace-plugin/blob/master/README.md>
- Integration via the Dynatrace APIs (Timeseries and Events)
- Based on a Performance Signature of a release (monspec)





# Install Jenkins Plugin

- Navigate to **Manage Jenkins -> Manage Plugins** page and switch to the Available tab. Search for the "Dynatrace" keyword, select the 3 options below and click "Install without restart".

Updates Available Installed Advanced

| Install ↓  | Name  | Version |
|--|---|---------|
| <input type="checkbox"/> <a href="#">Dynatrace Application Monitoring</a>            | Allows Dynatrace Application Monitoring test results to be published.                 | 2.1.3   |
| <input type="checkbox"/> <a href="#">Performance Signature: Dynatrace AppMon</a>     | This plugin collects Dynatrace Performance Signature Reports and stores them locally. | 3.1.4   |
| <input checked="" type="checkbox"/> <a href="#">Performance Signature: UI</a>        | This plugin collects Dynatrace Performance Signature Reports and stores them locally. | 3.1.4   |
| <input checked="" type="checkbox"/> <a href="#">Performance Signature: Viewer</a>    | This plugin collects Dynatrace Performance Signature Reports and stores them locally. | 3.1.4   |
| <input checked="" type="checkbox"/> <a href="#">Performance Signature: Dynatrace</a> | This plugin collects Dynatrace Performance Signature Reports and stores them locally. | 3.1.4   |

**Install without restart**   **Download now and install after restart**   Update information obtained: 1 hr 47 min ago   **Check now**



# Dynatrace API

- A Dynatrace API Token is needed for the Integration (Settings -> Integration -> Dynatrace API)
  - Only need the default top 3 permissions (This can be the same API Token as created in earlier steps)

The screenshot shows the Dynatrace Settings interface with the path: Settings > Integration > Dynatrace API. The main content area is titled "Dynatrace API" and contains instructions about using the API to export monitoring data. It features two tabs: "My Dynatrace API tokens" (which is selected) and "Other Dynatrace API tokens". Under the "My Dynatrace API tokens" tab, there is a button labeled "Generate token". Below this, a table lists existing tokens:

| Token name            | Owner | Actions   |
|-----------------------|-------|---|
| Performance Signature | admin | <span>Disable/enable</span> <span>Delete</span> <span>Edit</span> |

Below the table, there is a section for "Generated token" with a copy button. At the bottom, there is a new token creation form with a placeholder "Token name" set to "Performance Signature".



# Jenkins Configuration

- Navigate to Manage Jenkins -> Configure System -> Performance Signature: Dynatrace SaaS/Managed
- Enter name of the
  - Server (ex. Dynatrace Server)
  - Your Tenant URL (<https://xxxxxxxxx.live.dynatrace.com>) – SaaS, (<https://asdfa.Dynatrace-managed.com/e/asdfafa>) - Managed
  - API Token
  - Note: You many need to Save and exit this screen and come back for the "Add" dropdown to works, thanks to a bug in Jenkins
- Test the Connection and Save

Performance Signature: Dynatrace

Dynatrace server configurations

|                      | Dynatrace server  |
|----------------------|---|
| Name                 | Dynatrace Demo Environment  |
| Dynatrace Server URL | <input type="text" value="https://[REDACTED]/e/b060b9c0-d824-468a-8c7c-9df3816c815a/"/>     |
| Dynatrace API Token  | <input type="text" value="Dynatrace API token (Dynatrace API Token)"/> <button>Add</button> |

[Advanced...](#)

[Test connection](#)



## Adding Events to our Pipeline

---

- There are three main events we will be adding to our Deployment Pipeline:
  - `createDynatraceDeploymentEvent` – Sends the Deployment Event to Dynatrace
  - `recordDynatrace Session` – Sets start/end time for when Perf Test beings
  - `perfSigDynatraceReports` – Queries all the metrics during the perf test timeframe and compares against the spec file. If metrics violate the spec file, the pipeline stage is failed



# Adding Events to our Pipeline

---

- In the DeploySockShop pipeline, click “Configure” and scroll down to the Pipeline script

## Pipeline

Definition

Pipeline script

```
1> pipeline{
2>     agent {
3>         label 'jenkins-slave'
4>     }
5>     parameters {
6>         choice(name: 'BUILD', choices: ['One', 'Two'], description: 'Choose Build')
7>     }
8>
9>     stages{
10>         stage('Checkout code'){
11>             steps{
12>                 git 'https://github.com/dynatrace-acm/dtacmworkshop.git'
13>             }
14>         }
15>         stage('Build Code') {
16>             steps{
17>                 container('mvn'){
18>                     ...
19>                 }
20>             }
21>         }
22>     }
23> }
```

Use Groovy Sandbox

[Pipeline Syntax](#)



# Adding Events to our Pipeline – Deployment Events

- First we want to add a Deployment Event to our Pipeline so every time we deploy our build to staging, Dynatrace will know about it

```
stage('Deploy to Staging'){
    steps{
        container('kubectl'){
            echo 'Deployment canary build...'
            createDynatraceDeploymentEvent(envId: 'Dynatrace Server',
                tagMatchRules: [[meTypes: [[meType: 'SERVICE']]],
                tags: [[context: 'KUBERNETES', key: 'app', value: 'carts'],
                    [context: 'KUBERNETES', key: 'stage', value: 'dev']]]) {
                // some block
            }
            script{
                if (params.BUILD == "One") {
                    sh 'ls $WORKSPACE'
                    sh 'kubectl apply -f $WORKSPACE/manifests/sockshop-app/dev/carts2.yml'
                    echo "Waiting for carts service to start..."
                    sleep 350
                } else {
                    sh 'ls $WORKSPACE'
                    sh 'kubectl apply -f $WORKSPACE/manifests/sockshop-app/canary/carts2-canary.yml'
                    echo "Waiting for carts service to start..."
                    sleep 350
                }
            }
        }
    }
}
```



# Adding Events to our Pipeline – Record Perf Test

- Next we want to wrap our Loadtest stage with the recordDynatraceSession function, so we know when tests start and stop

```
stage ('Run Load Test'){
    steps{
        container('kubectl'){
            echo 'Get Carts URL'
            script{
                dir("loadtest"){
                    sh "chmod +x cartstest.sh"
                    sh "./cartstest.sh"

                }
            }
        }
        container('jmeter'){
            script{
                recordDynatraceSession(envId: 'Dynatrace Server',
                    tagMatchRules: [[meTypes: [[meType: 'SERVICE']]],
                    tags: [[context: 'KUBERNETES', key: 'app', value: 'carts'],
                        [context: 'KUBERNETES', key: 'stage', value: 'dev']])),
                    testCase: 'loadtest') {

                    if (params.BUILD == "One"){
                        dir("loadtest"){
                            sh "jmeter -n -t carts_load1.jmx -l testresults.jtl"
                        }
                    } else {
                        dir("loadtest"){
                            sh "jmeter -n -t carts_load2.jmx -l testresults.jtl"
                        }
                    }
                }
            }
        }
    }
}
```



## Adding Events to our Pipeline – Compare results to monspec

---

- After the load test is completed, we want to validate the test against our monspec file (see next slide), to determine if our build meets quality or not
- Add the below stage ("Dynatrace Quality Gate") to the very end of our pipeline

```
stage ('Dynatrace Quality Gate'){
    steps{
        perfSigDynatraceReports envId: 'Dynatrace Server', nonFunctionalFailure: 2, specFile: 'monspec.json'
    }
}
```



## Sample Monspec File

---

```
{  
  "spec_version": "2.0",  
  "timeseries": [  
    {  
      "timeseriesId": "com.dynatrace.builtin:service.responsetime",  
      "aggregation": "avg",  
      "tags": "[Kubernetes]app:carts,[Kubernetes]stage:dev",  
      "upperWarning": 1000,  
      "upperSevere": 3000  
    }  
  ]  
}
```



## Adding Events to our Pipeline

---

- Once the events are added to our Pipeline we can apply and save
- Note: There is a sample Pipeline with the events already added in the root of the repository titled "PipelineWithIntegration.txt"
- You can use this as a reference when editing your pipeline
  - Note: You will need to make sure you change the name of the Dynatrace Environment to the same thing as you named your connection earlier. This file uses the name "Dynatrace Server" as the Dynatrace Environment Name.

# Let's test our Integration

---





## Deploy Build Two

- After updating and saving our Pipeline to include the Dynatrace Performance Events, let's re-run our pipeline with build Two

Jenkins

Jenkins > DeploySockShop >

Back to Dashboard

Status

Changes

Build with Parameters

Delete Pipeline

Configure

Full Stage View

Rename

Performance Signature

Pipeline Syntax

## Pipeline DeploySockShop

This build requires parameters:

BUILD Two ▾

Choose Build

Build

Select Build "Two" and Build



# Now our build should fail

## Stage View

|    | Checkout code   | Build Code | Build Image | Push Image to Repo | Deploy to Staging | Run Load Test | Dynatrace Quality Gate |
|----|-----------------|------------|-------------|--------------------|-------------------|---------------|------------------------|
|    |                 |            |             |                    |                   |               | 1s                     |
| #7 | Jul 19<br>11:28 | No Changes | 679ms       | 6s                 | 3s                | 3s            | 5min 58s               |
| #6 | Jul 19<br>11:16 | 1 commit   | 1s          | 7s                 | 3s                | 3s            | 5min 58s               |



# Deploy Ansible Tower

- If we redeploy using Build #1, this build should pass our Dynatrace Quality Gate

Jenkins ➤ DeploySockShop ➤

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Build with Parameters](#)

[Delete Pipeline](#)

[Configure](#)

[Full Stage View](#)

[Rename](#)

[Performance Signature](#)

[Pipeline Syntax](#)

## Pipeline DeploySockShop

This build requires parameters:

BUILD  Choose Build

Select Build “One” and Build

# Auto-remediation with Ansible Tower

---

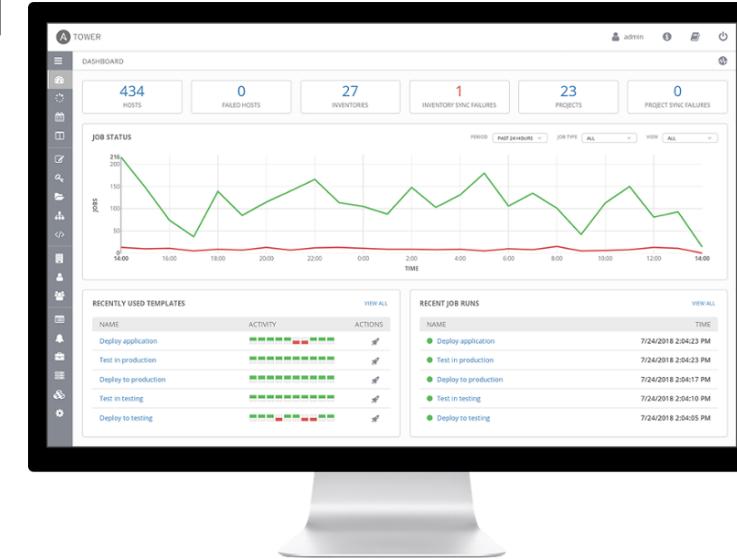


# Ansible and Ansible Tower

- Runbook – definition: a 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





## Promotional campaign

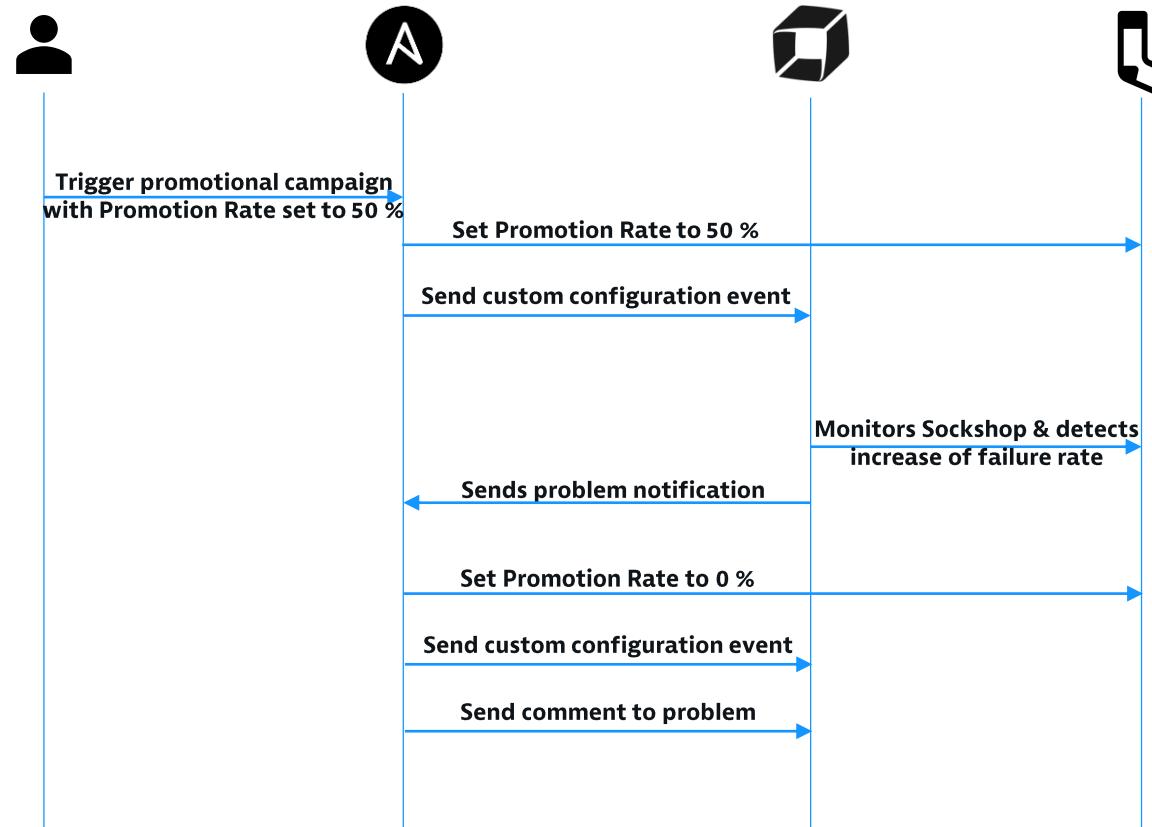
---

- We'll be implementing a promotional campaign for our socks
- This promotion will be randomly available to customers
- The carts service has an endpoint allowing us to provide the percentage of users that will benefit from the promotional gift
- The problem is that the promotion implementation is defective 😞
- Nobody knows about it yet and this could potentially have disastrous effect on the brand and business reputation





# Runbook automation workflow





# Deploy Ansible Tower

- In Terminal/Cloudshell execute the script: **deployTower.sh** in the directory **4-DeployTower**
- This will deploy and startup Ansible Tower. This will also configure Tower, import projects, inventories, credentials (DT API token) and playbook templates.
- Take a note of the Ansible Job URL displayed by the script. You will need it for a next step.

```
Dload Upload total spent left speed
100 3059 100 2896 100 163 4009 225 ---:--- ---:--- ---:--- 4005
REMEDIATION_TEMPLATE_ID: 35
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
100 3357 100 3053 100 304 3659 364 ---:--- ---:--- ---:--- 3656
STOP_CAMPAIGN_ID: 36
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
100 3395 100 3054 100 341 4553 508 ---:--- ---:--- ---:--- 4558
START_CAMPAIGN_ID: 37
-----
Ansible has been configured successfully! Copy the following URL to set it as an Ansible Job URL in the Dynatrace notification settings:
https://34.69.184.20/#/templates/job\_template/35
```



# Log in the Ansible Tower console

- Use the job template URL you copied earlier, without the path, and copy it in your browser to load the Ansible Tower console
  - For example, if the job template URL was :  
[https://34.69.184.20/#/templates/job\\_template/35](https://34.69.184.20/#/templates/job_template/35)  
then the console URL is : <https://34.69.184.20>
- You will be prompted to provide a license file and accept the license agreement
  - If you don't already have your license file, click the button to request a free trial license from Red Hat (delivered via email)
- Log in Ansible Tower
  - username : admin
  - password : dynatrace

**TOWER LICENSE**

Welcome to Ansible Tower! Please complete the steps below to acquire a license.

1 Please click the button below to visit Ansible's website to get a Tower license key.

**REQUEST LICENSE**

2 Choose your license file, agree to the End User License Agreement, and click submit.

\* LICENSE FILE

**BROWSE** No file selected.

\* END USER LICENSE AGREEMENT

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

This end user license agreement ("EULA") governs the use of the Ansible Tower software and any related updates, upgrades, versions, appearance, structure and organization (the "Ansible Tower Software"), regardless of the delivery mechanism.

1. License Grant. Subject to the terms of this EULA, Red Hat, Inc. and its affiliates ("Red Hat") grant to you ("You") a non-exclusive, non-transferable, limited right to use the Ansible Tower Software for your internal business purposes.

I agree to the End User License Agreement

By default, Tower collects and transmits analytics data on Tower usage to Red Hat. This data is used to enhance future releases of the Tower Software and help streamline customer experience and success. For more information, see [this Tower documentation page](#). Uncheck this box to disable this feature.

**SUBMIT**



# Configure Dynatrace Problem Notification

- In the Dynatrace UI, navigate to Settings -> Integration -> Problem notification -> Ansible Tower

The screenshot shows the Dynatrace UI with the following navigation path: Create custom chart > Settings > Integration > Problem notifications > Set up notifications.

The main content area is titled "Integrate with other notification systems". It contains a list of integration options:

- OpsGenie
- VictorOps
- PagerDuty
- Slack
- HipChat
- ServiceNow
- Ansible Tower (highlighted with a red box)
- JIRA
- Trello
- xMatters
- Email (highlighted with a red arrow)
- Custom integration

A red box highlights the "Ansible Tower" entry, and a red arrow points to the "Email" entry.



# Configure Ansible Tower integration

- Enter the Ansible Tower job template URL you were provided after the deployment
- Enter the credentials to access Ansible
  - username : admin
  - password : Dynatrace
- Click the Sent test notification button to validate your configuration
  - on success, a green confirmation message will be displayed
- Save your configuration

AnsibleTower integration test successful  
You can now save the configuration.

Edit Ansible Tower integration

Name your Ansible Tower integration

Remediation Playbook

Ansible Tower job template URL

`https://34.69.184.20/#/templates/job_template/35`

For example, `HTTPS://<Ansible Tower server name>/#/templates/job_template/<JobTemplateID>`

Note: Be sure to select the **Prompt on Launch** option in the Extra Variables section of your job template configuration.

Accept any SSL certificate (including self-signed and invalid certificates)

Username

admin

Password

.....

[Change password](#)

Job template ID

35

Custom Message (optional)

This message will be displayed in the Extra Variables **Message** field of your job template.

Alerting profile

Default

Select an [alerting profile](#) to control the delivery of problem notifications related to this integration.

[Send test notification](#) [Save](#) [Cancel](#)



## Adjust Anomaly Detection

Both problem and anomaly detection in Dynatrace leverage AI technology. This means that the AI learns how each and every microservice behaves and baselines them. Therefore, in a demo scenario like we have right now, we have to override the AI engine with user-defined values to allow the creation of problems due to an artificial increase of a failure rate. (**Please note:** if we would have the application running and simulate end-user traffic for a couple of hours/days there would be no need for this step.)

In your Dynatrace tenant, navigate to “Transaction & services” and filter by:  
[Kubernetes]app:carts and [Kubernetes]environment:production

The screenshot shows the Dynatrace interface for "Transaction & services". At the top, there are two filters: "Tag: [Kubernetes]app:carts" and "Tag: [Kubernetes]stage:prod". On the left, there are three filter dropdowns: "Problem impact" set to "any", "Service type" set to "any", and "Technology" set to "any". The main area displays "3 Services" with a table. The first service listed is "CartsController" (SpringBoot carts works.weave.socks.cart.CartApplication carts-\*). The second service listed is "ItemsController" (SpringBoot carts works.weave.socks.cart.CartApplication carts-\*), which is highlighted with a red rectangular border around its name and technology information.



## Adjust Anomaly Detection (Cont.)

- Click on the **ItemsController** and then on the three dots ( ...) next to the service name. Click on *Edit*

The screenshot shows the Dynatrace 'Transactions & services' interface. The top navigation bar is blue with the text 'Transactions & services' and 'ItemsController'. The main area displays the 'ItemsController' service details. It includes a circular icon with a leaf, the service name 'ItemsController', and a timestamp 'Last call 47 seconds ago'. Below this is a section titled 'Properties and tags' with several Kubernetes labels: [Kubernetes]app: carts, [Kubernetes]pod-template-hash: 6469cbcfcf77, [Kubernetes]product: s, [Kubernetes]release: stable, and 3 more...'. To the right of these properties is a context menu with options: 'Edit', 'Pin to dashboard', 'Process crash details', and 'Memory dump details'. Below the properties are three blue boxes: '0 Applications', '0 Services', and '1 Network client'. In the center, there is a throughput chart showing '12 /min Throughput' with a histogram. To the right of the throughput chart is another blue box: '0 Services' and '1 Database'. The overall background is light gray.



## Adjust Anomaly Detection (Cont.)

On the next screen, edit the anomaly detection settings as seen in the following screenshot. - **Global anomaly detection** has to be **turned off** - Detect increases in **failure rate** using **fixed thresholds** - Alert if **0 %** custom failure rate threshold is exceed during any 5-minute period. - Sensitivity: **High**

Transactions & services > ItemsController > Settings > Anomaly detection

**Service settings**  
ItemsController

**Anomaly detection**

Use global anomaly detection [settings](#)

Detect response time degradations automatically

Alert if the response time degrades by more than  ms and by  %.

Alert if the response time of the slowest 10% degrades by more than  ms and by  %.

To avoid over-alerting do not alert for low load services with less than  requests/min.

Detect increases in failure rate using fixed thresholds

Alert if  % custom failure rate threshold is exceeded during any 5-minute period.

Sensitivity:



# Launch Campaign playbook

- Navigate back to the Ansible Tower UI
- From the side menu, navigate to Resources -> Templates
- Click on the rocket icon to launch the start-campaign playbook
- Hit Next on the prompt popup window and then Launch

**start-campaign** Job Template

| ACTIVITY      | inventory                     |
|---------------|-------------------------------|
| PROJECT       | self-healing                  |
| CREDENTIALS   | mcl26608 API token            |
| LAST MODIFIED | 9/23/2019 7:42:21 PM by admin |
| LAST RAN      | 9/23/2019 7:42:21 PM          |

Start a job using this template

ate that the recap shows OK=2

```
5 PLAY [localhost] ****
6
7 TASK [update percentage of promotion for carts service] ****
8 ok: [localhost]
9
10 TASK [update promotion configuration for carts service] ****
11 ok: [localhost]
12
13 PLAY RECAP ****
14 localhost : ok=2    changed=0    unreachable=0    failed=0
15
```



# Observe the promotional campaign

- In the Dynatrace console, navigate to Transactions & Services and drill-down into the production ItemsController service
  - Tip : filter by tag : [Kubernetes]stage:prod to only display services in production

The screenshot shows the Dynatrace Services page with the following details:

Services

Filtered by Tag: [Kubernetes]stage:prod

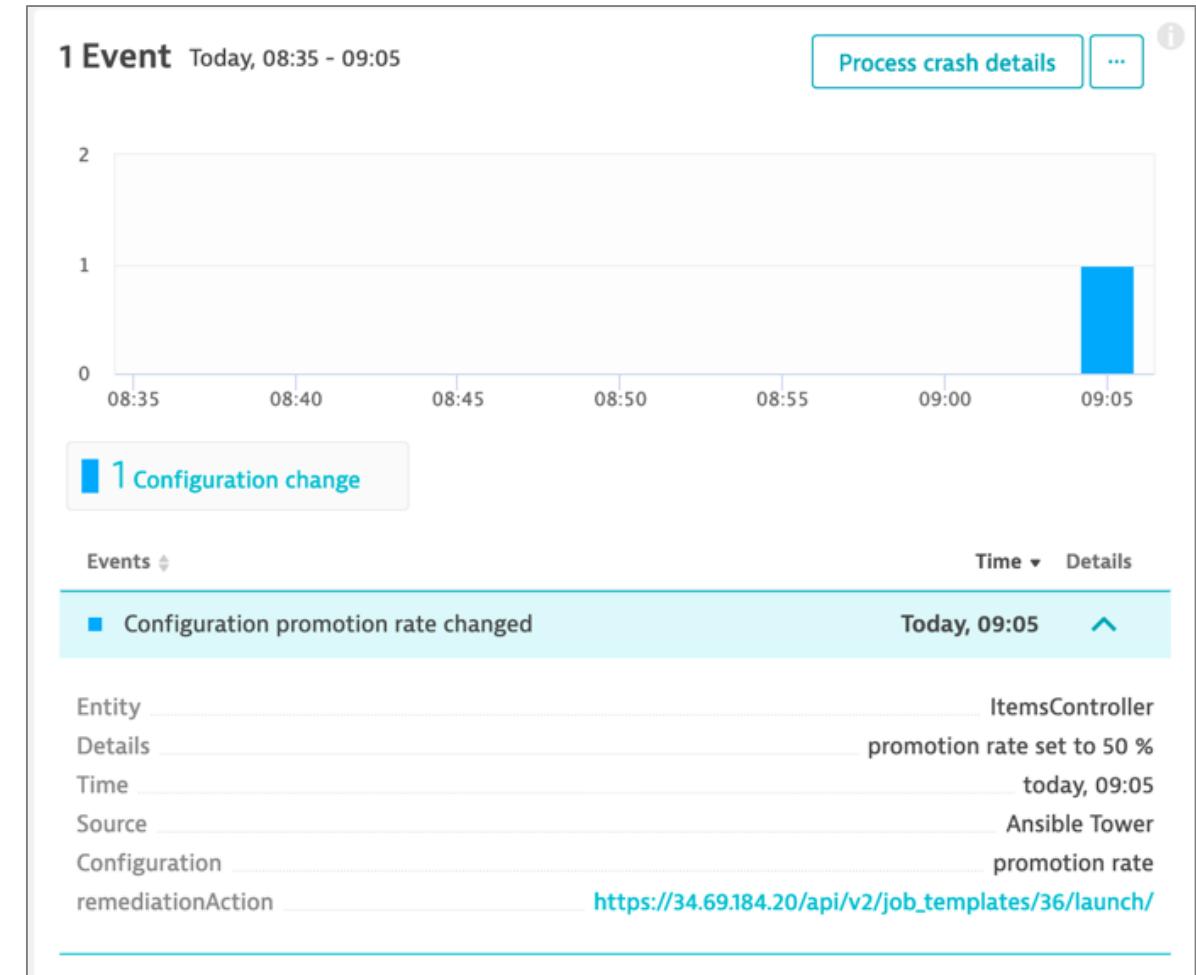
9 Services

| Name   | Response time median | Failure rate | Requests |
|--|----------------------|--------------|----------|
| ItemsController<br>SpringBoot carts works.weave.socks.cart.CartApplication carts-*                         | 7.99 ms              | 0 %          | 34 /min  |
| CartsController<br>SpringBoot carts works.weave.socks.cart.CartApplication carts-*                         | 3.28 ms              | 0 %          | 23 /min  |
| :8080<br>payment payment-*   | 276 µs               | 0 %          | 10 /min  |
| :8080<br>catalogue catalogue-*   | 605 µs               | 0 %          | 10 /min  |
| HealthCheckController<br>SpringBoot works.weave.socks.queueMaster.QueueMasterApplication queue-master-*    | 1.13 ms              | 0 %          | 10 /min  |
| :8080<br>user user-*   | 1.57 ms              | 0 %          | 9 /min   |
| HealthCheckController<br>SpringBoot orders works.weave.socks.orders.OrderApplication orders-*              | 3.96 ms              | 0 %          | 9 /min   |
| ShippingController<br>SpringBoot shipping works.weave.socks.shipping.ShippingServiceApplication shipping-* | 1.03 ms              | 0 %          | 9 /min   |
| front-end<br>server.js (front-end) front-end.stable-*  | 1.9 ms               | 0 %          | 7 /min   |



## Observe the promotional campaign

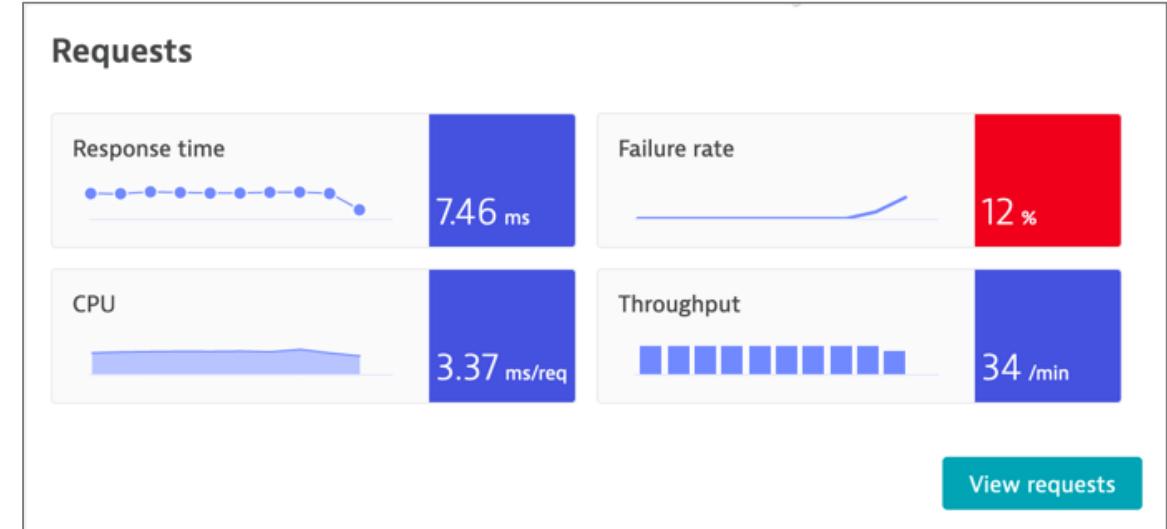
- Look at the Service events.
- The Start Campaign playbook have notified Dynatrace when the promotional rate was changed to 50%





# Observe the promotional campaign

- You should see the Failure Rate increasing, eventually leading to Dynatrace detecting a Problem
  - You might need to refresh your browser a few times



**⚠ Problem 575: Failure rate increase**

ItemsController  
Since 2019 Sep 25 09:01:00 (8 minutes)



# Observe the auto-remediation playbook actions

- Go back to the Service view
- The comments section will show the remediation actions taken by Ansible Tower

1 impacted service  
66.8 Requests per minute impacted

 ItemsController  
Web service

**Failure rate increase**  
by a failure rate increase to 18 %

Affected requests      Service method  
66.8 /min      2 Service methods

2 comments i

Invoked remediation action successfully executed.  
Ansible Playbook via Ansible Tower, Sep 25, 2019 09:07 Delete

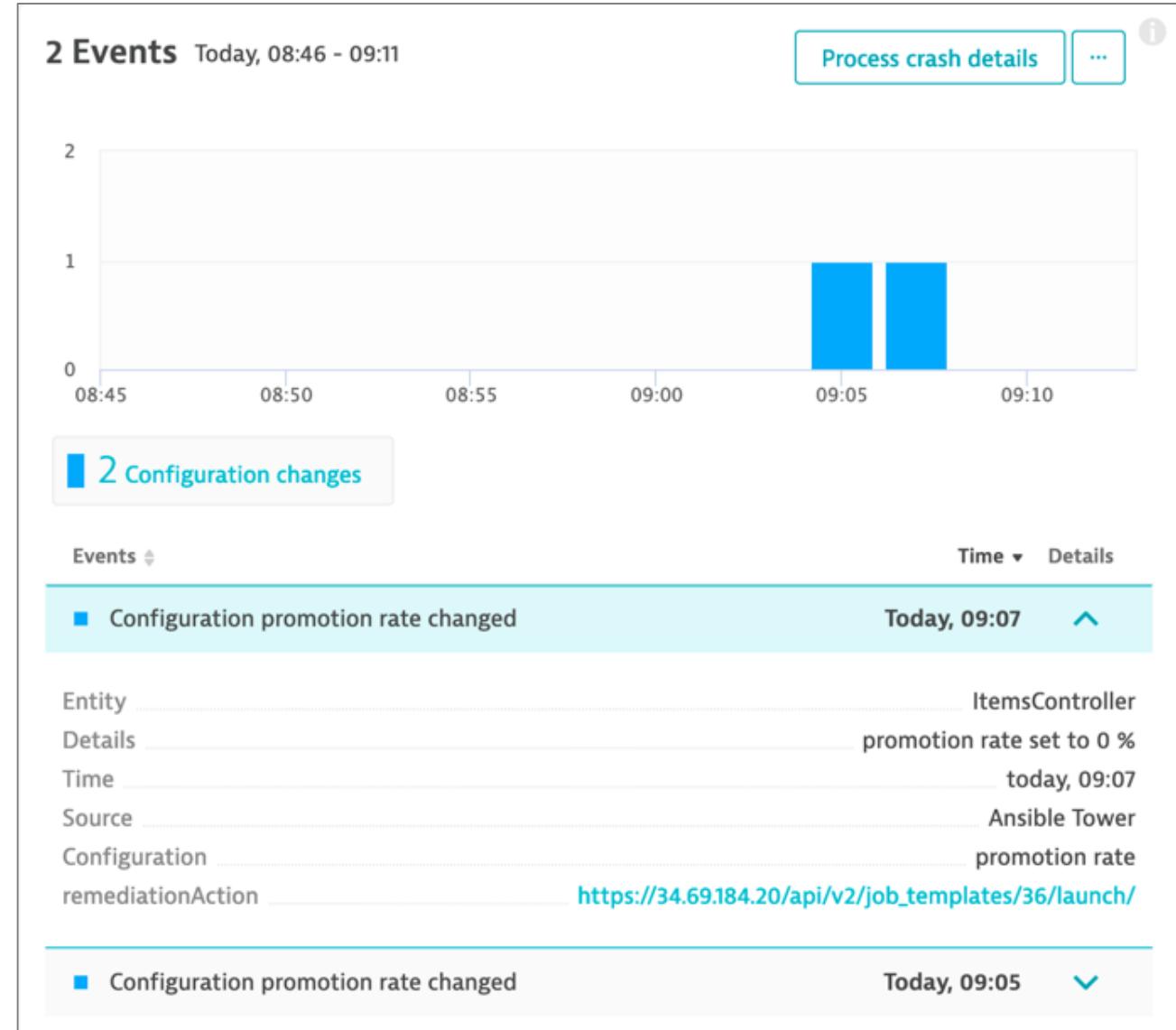
Remediation playbook started.  
Ansible Playbook via Ansible Tower, Sep 25, 2019 09:07 Delete

Add comment



# Observe the auto-remediation playbook actions

- Drill-down in the Problem.
- You will see a new configuration change event reported by Ansible Tower
- The promotional rate has been set back to 0% to remediate to the transaction failures





# Observe the auto-remediation playbook actions

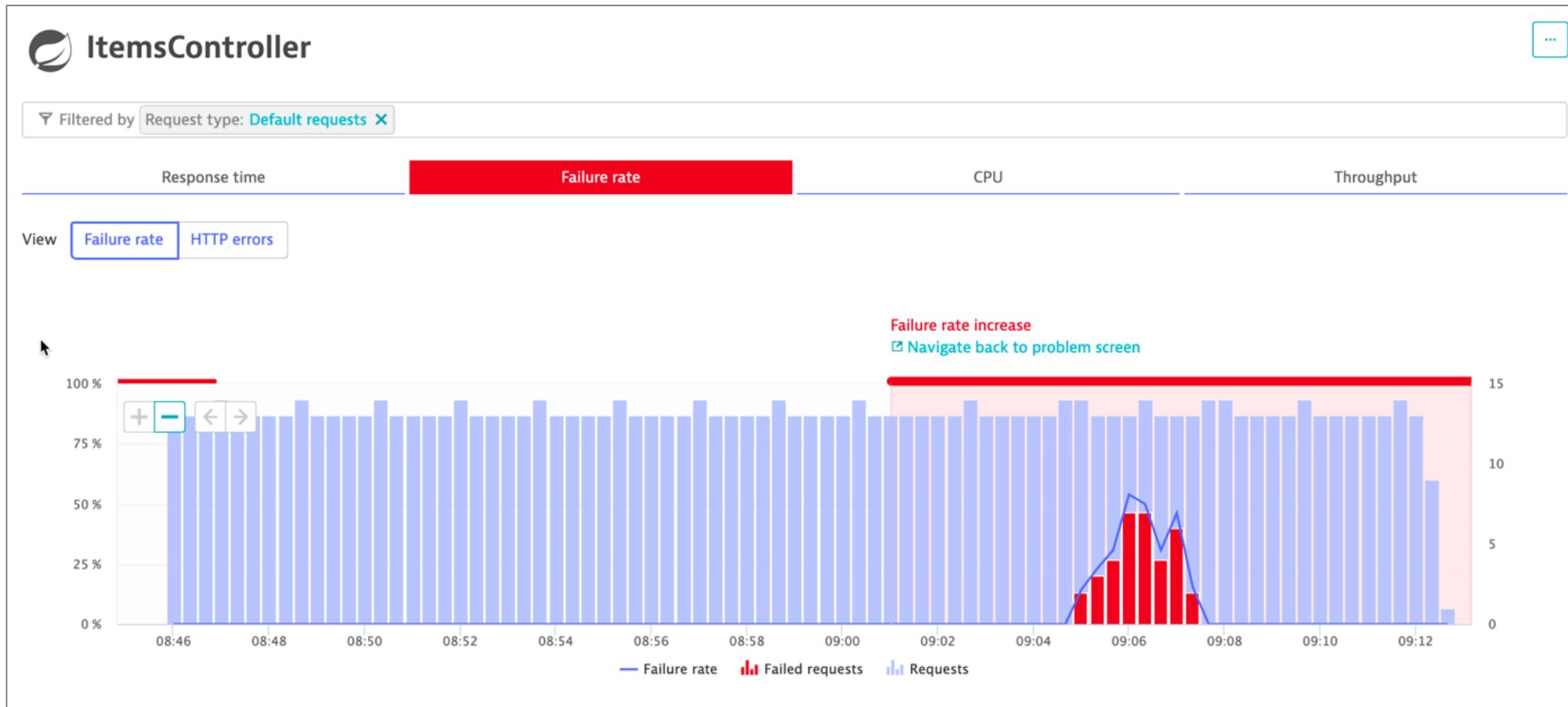
- Jobs executed in Ansible Tower
  - start-campaign (set rate to 50%)
  - remediation
    - push comment to Dynatrace Problem
    - retrieve problem details
    - launch remediation action related to problem context
    - update Dynatrace Problem
  - stop-campaign (set rate to 0%)

|   |
|---|
| <p>● 37 - stop-campaign <span style="border: 1px solid #ccc; padding: 2px;">Playbook Run</span></p> <p>STARTED 9/25/2019 9:07:33 AM FINISHED 9/25/2019 9:07:42 AM</p> <p>LAUNCHED BY admin</p> <p>JOB TEMPLATE stop-campaign</p> <p>INVENTORY inventory</p> <p>PROJECT self-healing</p> <p>CREDENTIALS <span style="background-color: #007bff; color: white; border-radius: 50%; padding: 2px 10px; border: 1px solid #007bff;">mcl26608 API token</span></p>   |
| <p>● 35 - remediation <span style="border: 1px solid #ccc; padding: 2px;">Playbook Run</span></p> <p>STARTED 9/25/2019 9:07:21 AM FINISHED 9/25/2019 9:07:33 AM</p> <p>LAUNCHED BY admin</p> <p>JOB TEMPLATE remediation</p> <p>INVENTORY inventory</p> <p>PROJECT self-healing</p> <p>CREDENTIALS <span style="background-color: #007bff; color: white; border-radius: 50%; padding: 2px 10px; border: 1px solid #007bff;">mcl26608 API token</span></p>       |
| <p>● 33 - start-campaign <span style="border: 1px solid #ccc; padding: 2px;">Playbook Run</span></p> <p>STARTED 9/25/2019 9:05:08 AM FINISHED 9/25/2019 9:05:17 AM</p> <p>LAUNCHED BY admin</p> <p>JOB TEMPLATE start-campaign</p> <p>INVENTORY inventory</p> <p>PROJECT self-healing</p> <p>CREDENTIALS <span style="background-color: #007bff; color: white; border-radius: 50%; padding: 2px 10px; border: 1px solid #007bff;">mcl26608 API token</span></p> |



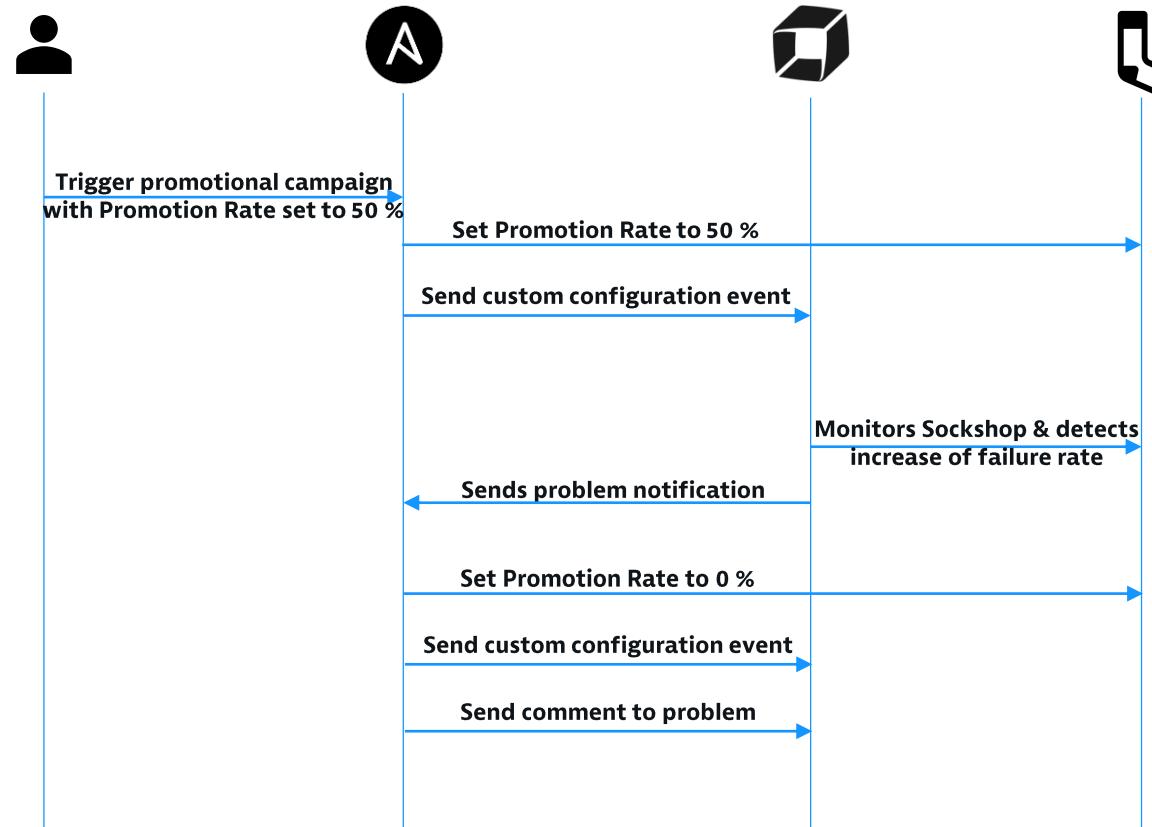
# That's it! Problem solved automatically

- Look at the Service Failure Rate. It has returned to 0%





# Runbook automation workflow recap



# Clean up

---



## Clean up

---

- If you want to keep the cluster running but:
  - remove Jenkins : execute the script **removeJenkins.sh** contained in the **utils** directory
  - remove Ansible Tower : execute the script **removeTower.sh** container in the **utils** directory
- If you want to delete everything, feel free to delete the GKE cluster via the GKE Console



[dynatrace.com](https://www.dynatrace.com)