Dynatrace aws Hands-On Workshop

Unbreakable Continuous Delivery

dynatrace

Your App / Container

Pivotal Cloud Foundry

OPENSHIFT

kubernetes

aws

Google Cloud Platform

Azure

SAP S/4 HANA

Cloud Native

Proof – autonomous cloud survey (median vs 95th percentile)

Verdict: The majority are not "cloud native" (yet)

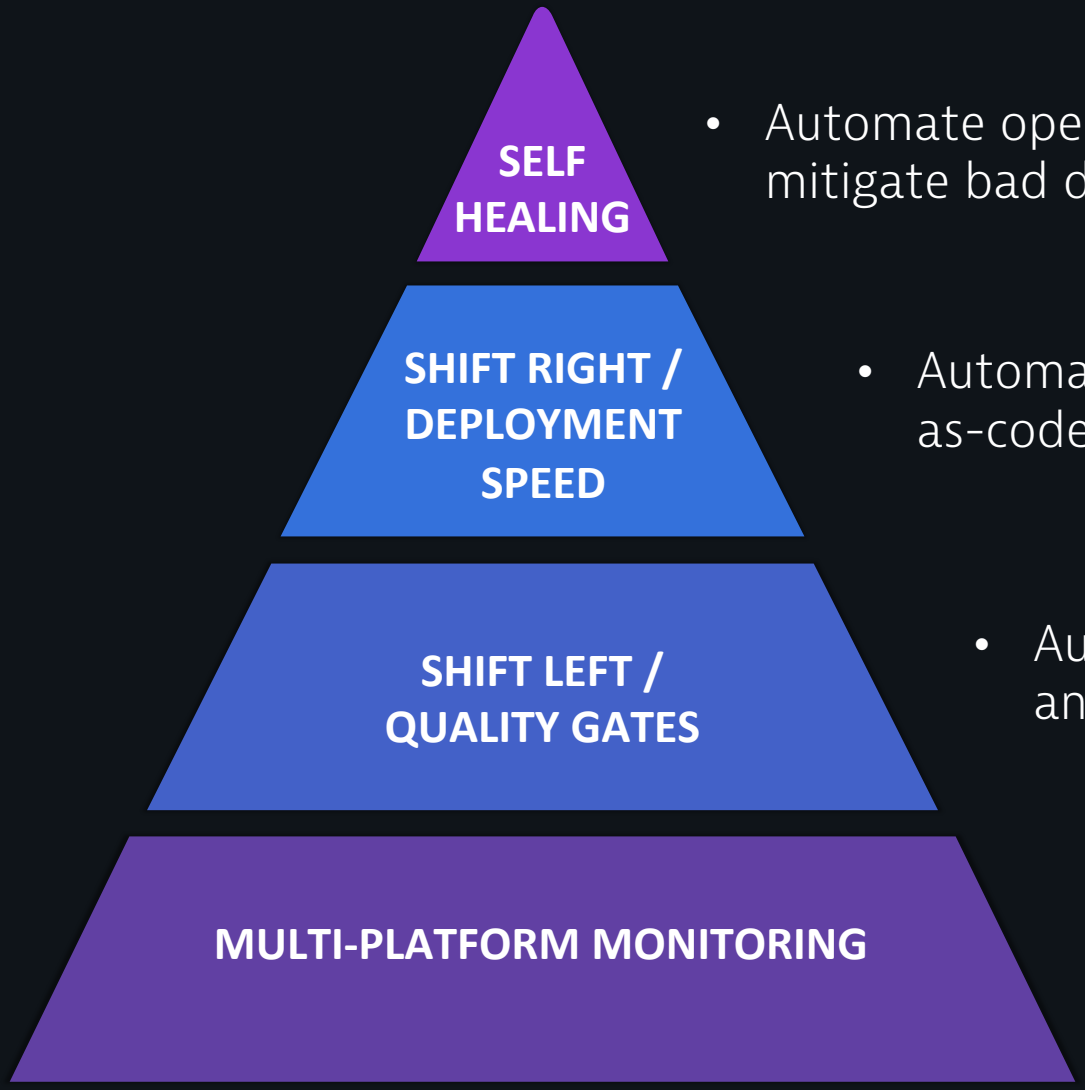
95th Percentile

2 days	1 out of 10	0 hotfixes	~4 hours

Median

2.5 weeks	3 out of 10	3 hotfixes	4.8 days
Code to Production (Commit Cycle Time)	Business **Impacting** Deployments	**Per** Production **Deployment**	**MTTR** (Mean Time to Repair)

Key blue print – unbreakable continuous delivery model



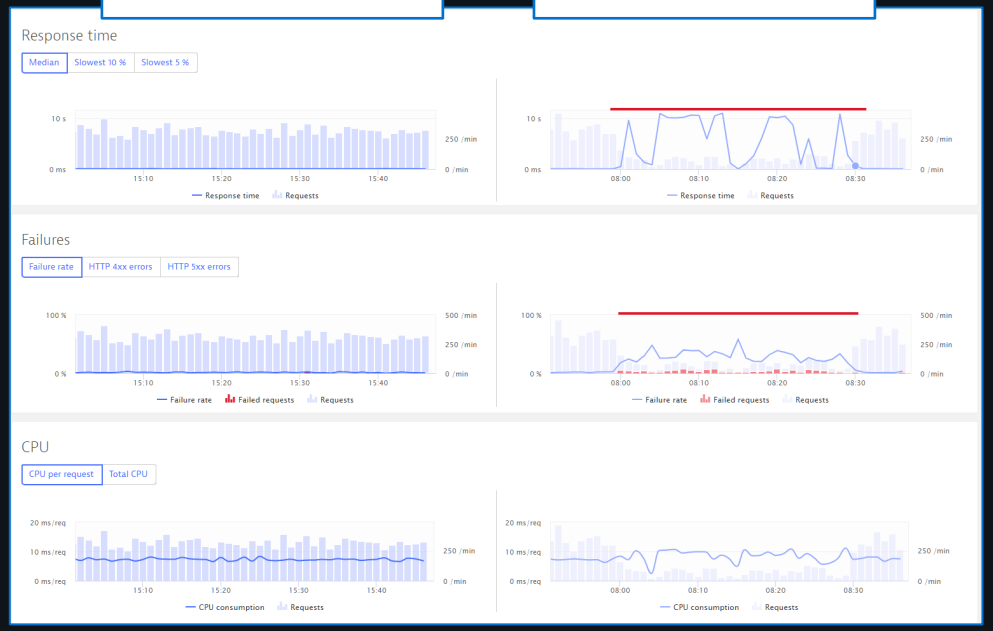
- Automate operations (self-healing) – auto-mitigate bad deployments in production
- Automate deployment (shift-right) – push "monitoring-as-code" for auto-validation and auto-alerting
- Automate quality (shift-left) – automate the pipeline and stop bad code changes before they reach prod
- Automate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

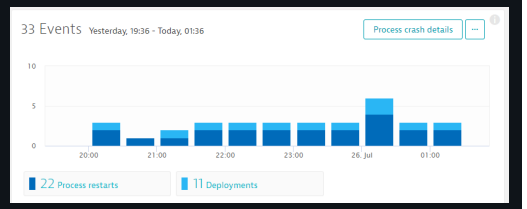
Unbreakable delivery pipeline in action

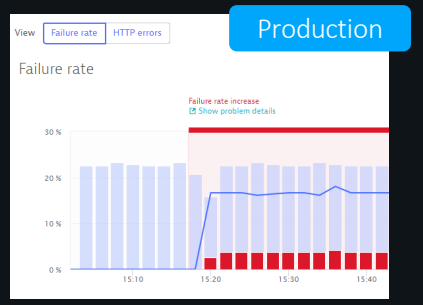
1 Staging → 2 Approve Staging → 3 Production → 4 Approve Production → 5

Push Context → Auto-Quality Gate → Push Context → Auto-Validate

Build #17 | Build #18

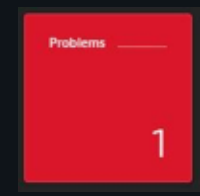




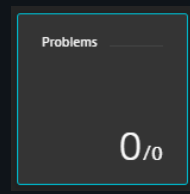


Deployment on CheckDestination 2018-01-23 09:00

Entity	CheckDestination
Time	today, 09:00 - 09:00 (-)
Approver	Alice McBright (alice.mcbright@easytravel.com)
Build Number	1.223.23432
CI	http://a-tower.local/job/DeployJob/38/artifact/build/Deployment-v1.23.321.zip
Git commit	e5a6baac7eb
Owner	Jason Miller (jason.miller@easytravel.com)
Project	CheckDestination
Remediation	http://a-tower.local/job/RemediateJob/38/
Source	ServiceNow
Version	1.23.321



Auto-Remediate!



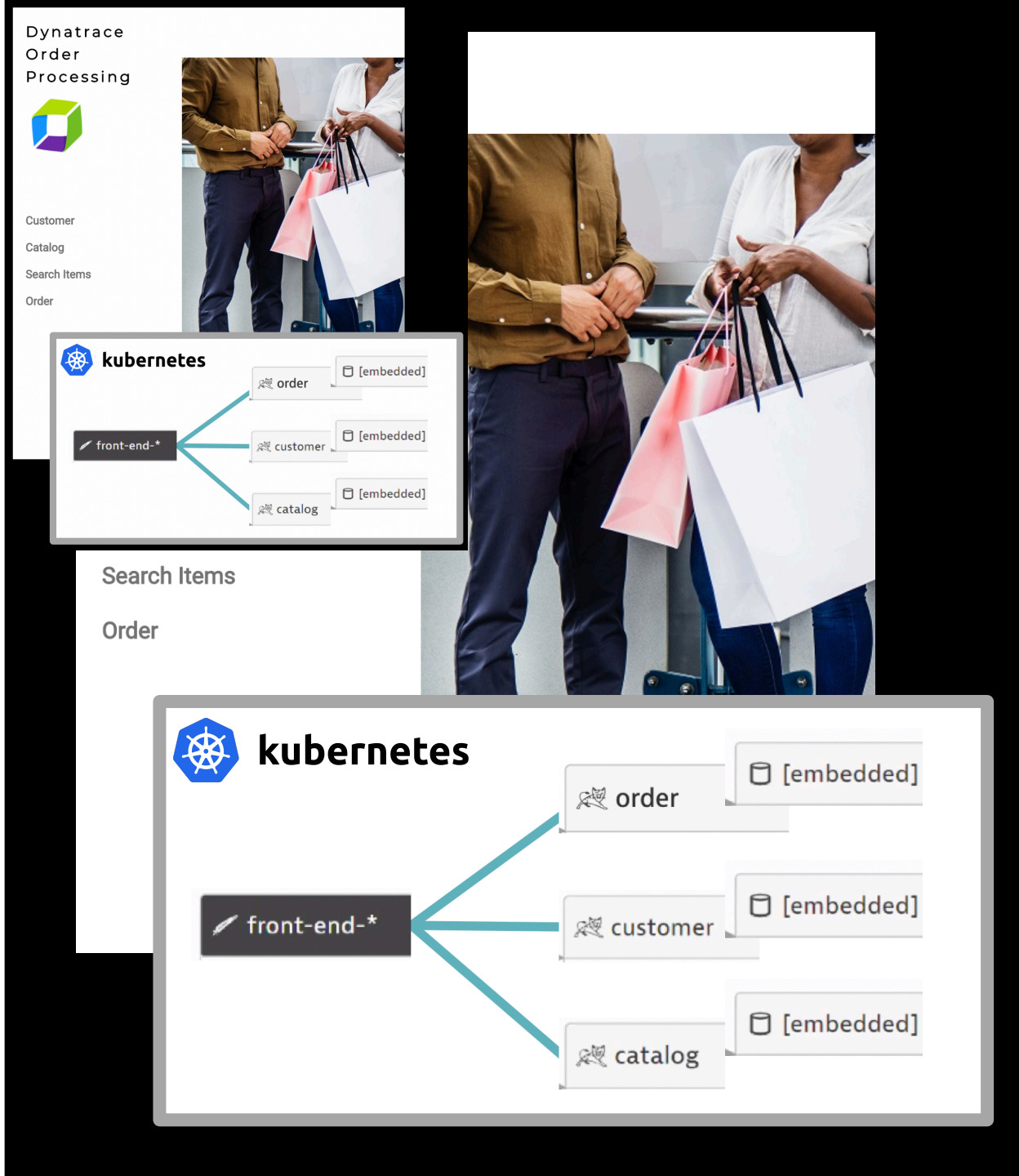
Show of hands

- Your functional area?
  - Development
  - Operations
  - SRE
  - DevOps
- (a) I can google it, (b) done some reading, (c) basic hands-on, (d) people come to me for help
  - Kubernetes
  - AWS – VPC, IAM, EC2, …
  - Docker
  - Dynatrace
  - Terraform

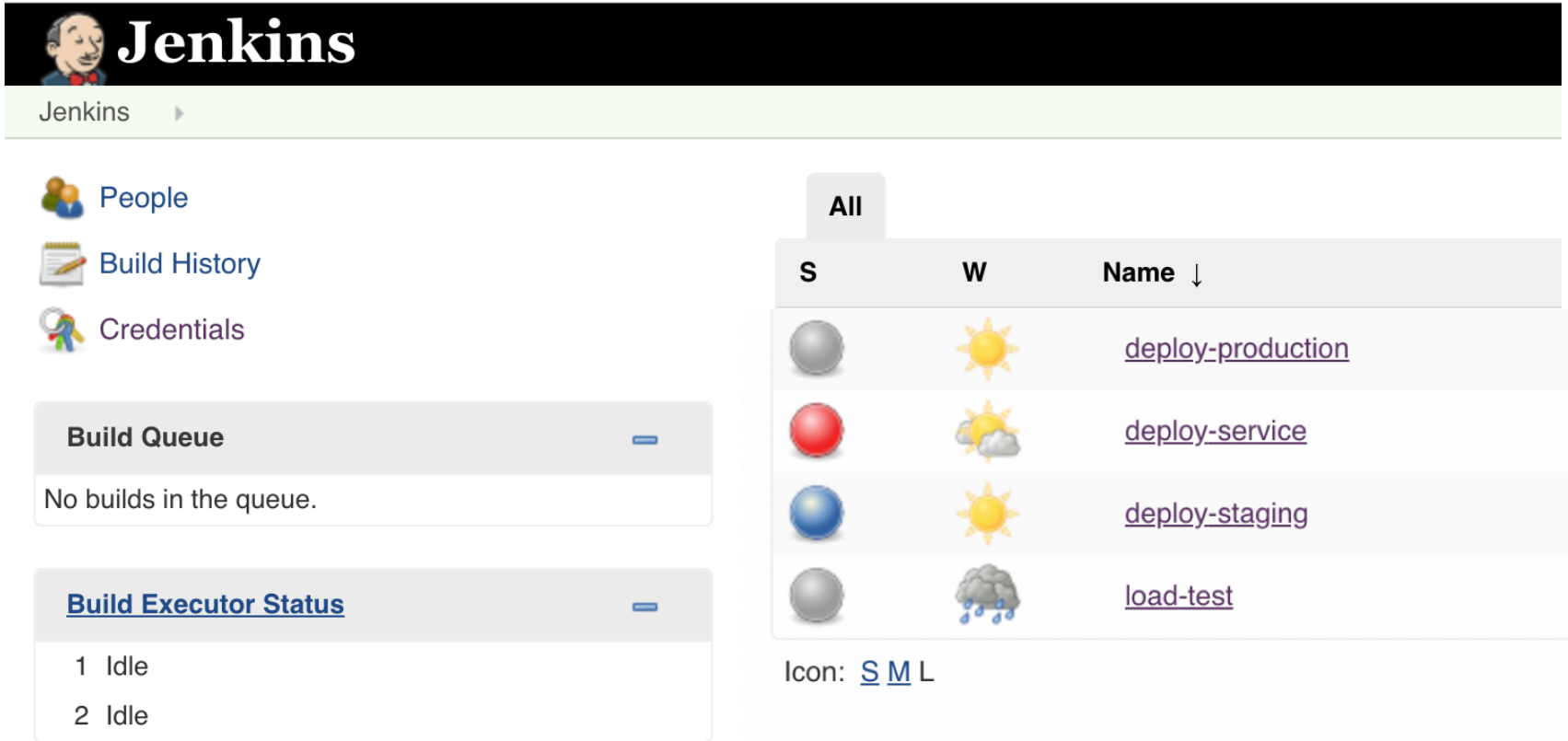
What we have prepared for you today…

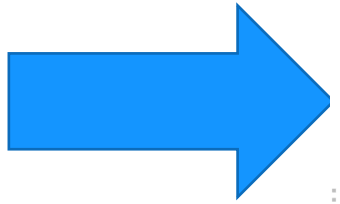
Workshop Application

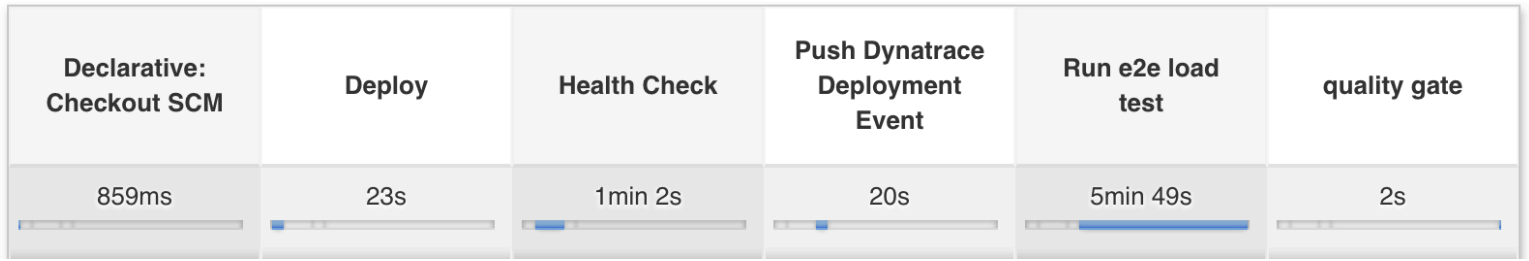
- Order processing application
- Web UI with 3 java spring boot microservices with embedded databases
- Components pre-compiled, built as a Docker image, and staged on github.com
- Application is deployed using Kubernetes deployment and services manifest files



Jenkins Pipelines to deploy and run performance tests





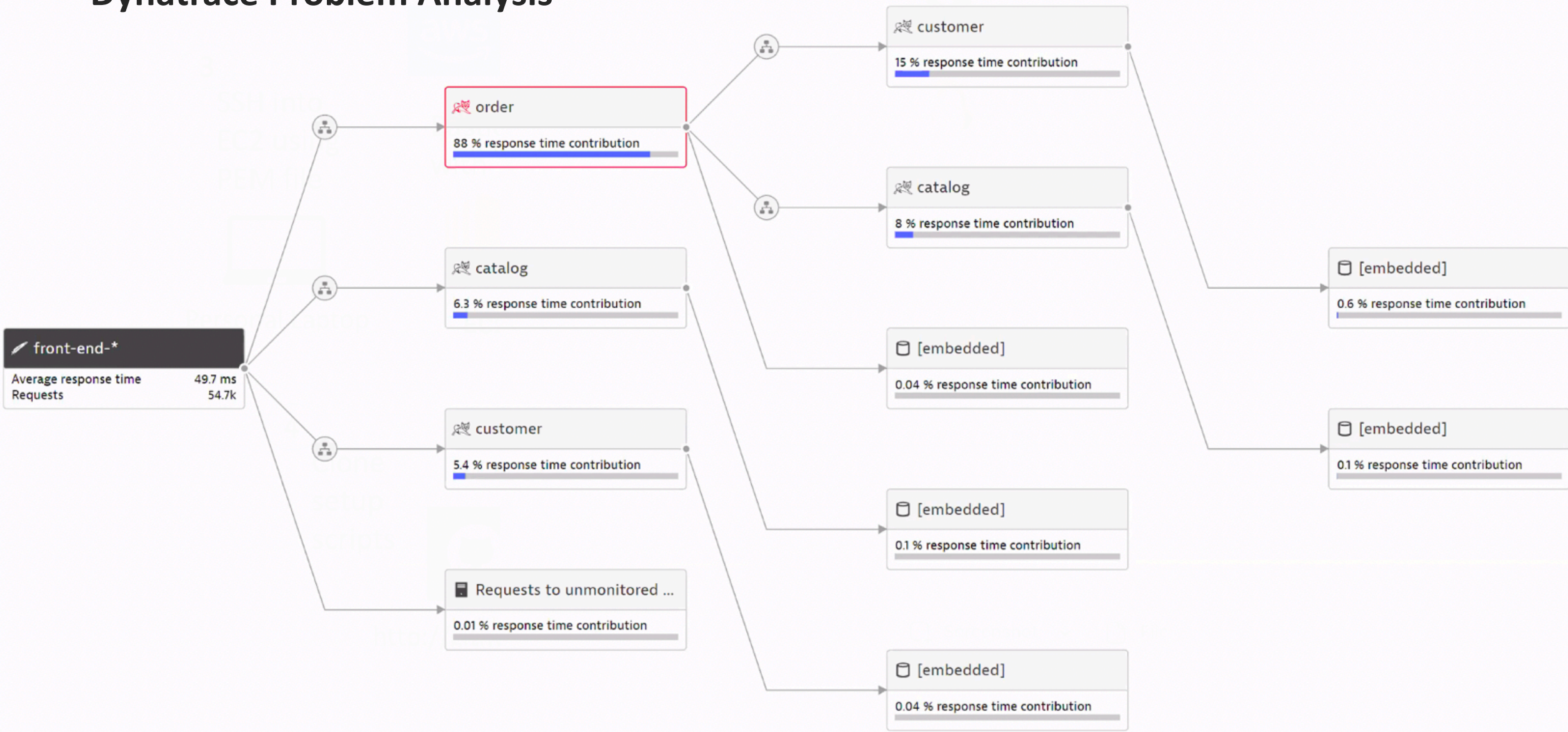


Quality Gates: T-Systems Performance Signature Jenkins Plugin

build #	PDF	date	Services - Error detection - order (Average) (Percent)	Services - Response time - customer (Maximum) (MilliSecond)
5		Mar 26, 2019 1:37:11 AM	0	89
4		Mar 26, 2019 1:25:04 AM	33.19	1832
3		Mar 26, 2019 1:14:28 AM	24.05	5009
2		Mar 26, 2019 12:54:13 AM	35.63	6629
1		Mar 26, 2019 12:23:29 AM	0	2295

https://github.com/T-Systems-MMS/perfsig-jenkins

Dynatrace Problem Analysis



Dynatrace AI Problem Detection

order: Multiple service problems

Problem 886 detected at 15:38 (open for 7 minutes).

Affected applications	Affected services	Affected infrastructure
–	1	1

Business impact analysis

An analysis of all affected service calls and impacted real users during the first 10 minutes of the problem shows the following potential impact.

0	270k
Impacted users	Affected service calls

Show more

1 impacted service

138 Requests per minute impacted

order
Web request service

Response time degradation

The current response time (1.37 s) within the slowest 10% exceeds the auto-detected baseline (53.8 ms) by 2,449 %

Affected requests	Service method
69 /min	All dynamic requests

Failure rate increase

by a failure rate increase to 9.86 %

Affected requests	Service method
69 /min	All dynamic requests

Root cause

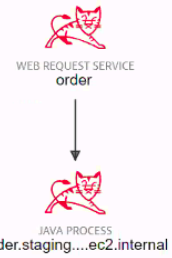
Based on our dependency analysis all incidents have the same root cause

order.staging (order-9689cf5d8-2w56f)
Process

Service process unavailable

Process order.staging (order-9689cf5d8-2w56f) on host ip-172-30-119-227.ec2.internal has been shut dow

Visual resolution path
Click to see how we figured this out.

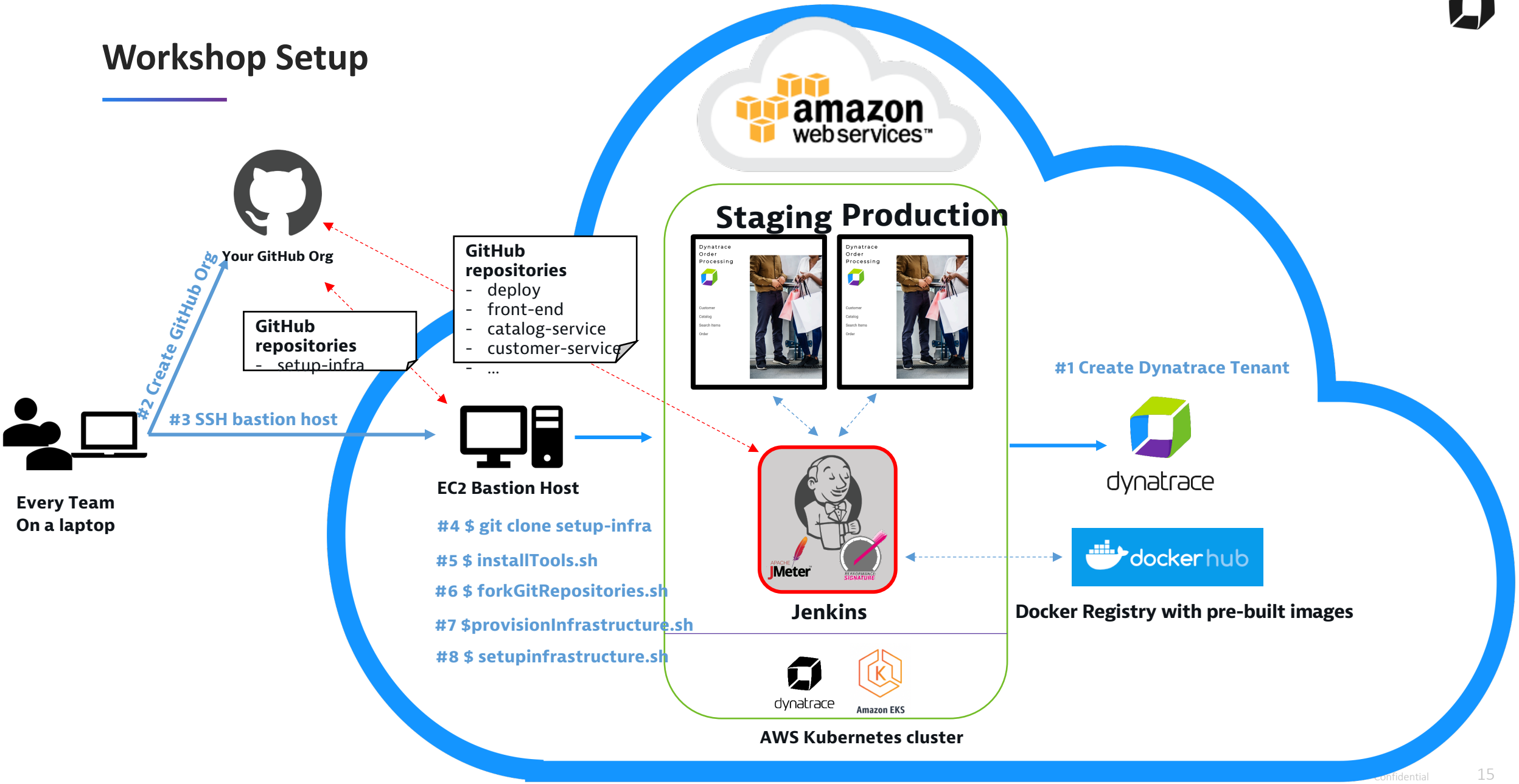


Agenda

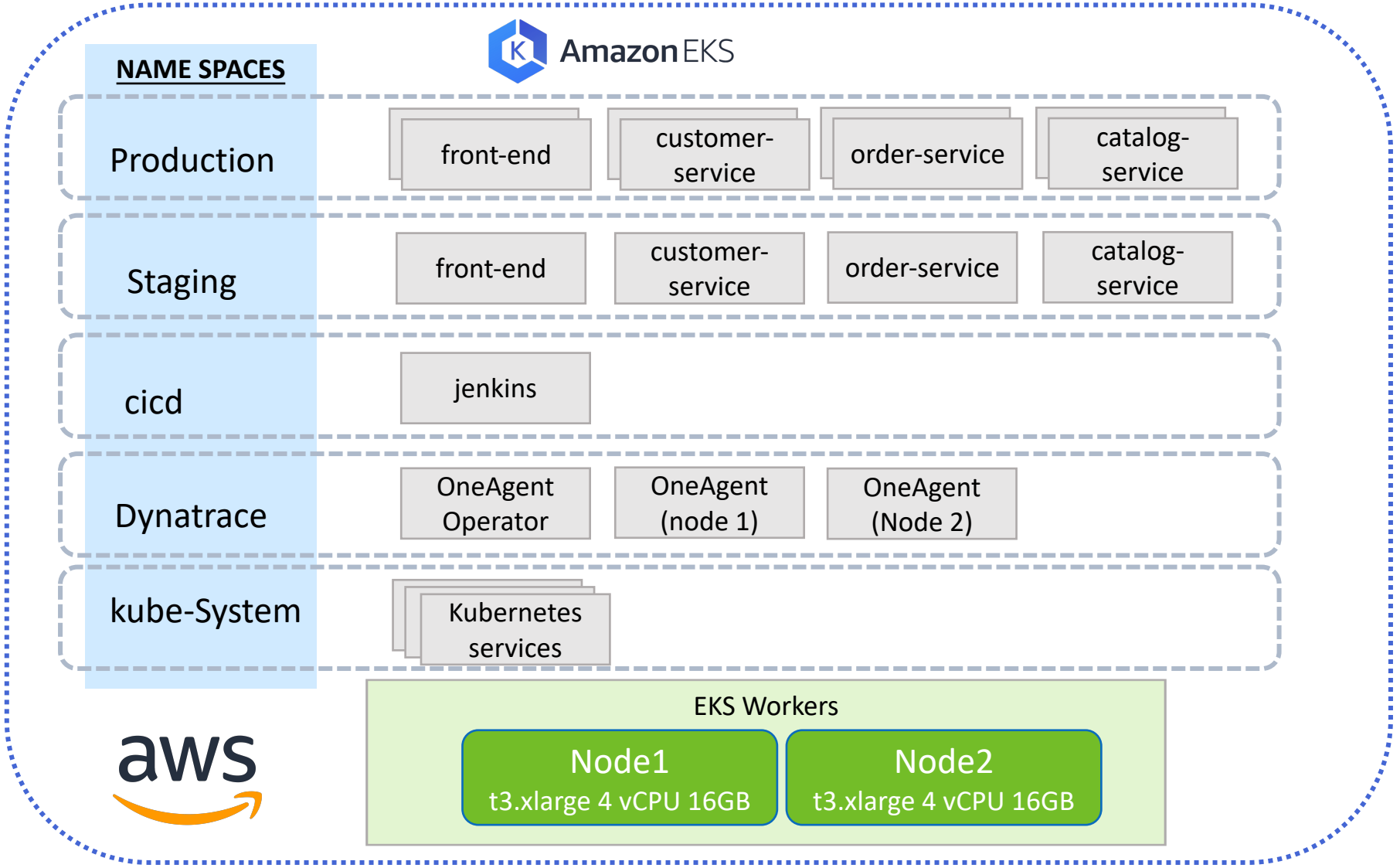
1. Demo infrastructure setup
   - *Overview*
   - *Setup EC2 Bastion host*
   - *Provision Infrastructure*
2. Concepts
   - *Automated Quality Gates (Shift-Left)*
   - *Automated Operations (Self-Healing)*
   - *Automated Deployment (Shift-Right)*
3. Workshop
   - *Deploy the application*
   - *Explore & Finish Dynatrace Configuration*
   - *Quality Gate*
   - *Self healing*
4. Recap

Demo infrastructure setup

Workshop Setup



Workshop Kubernetes Cluster Setup



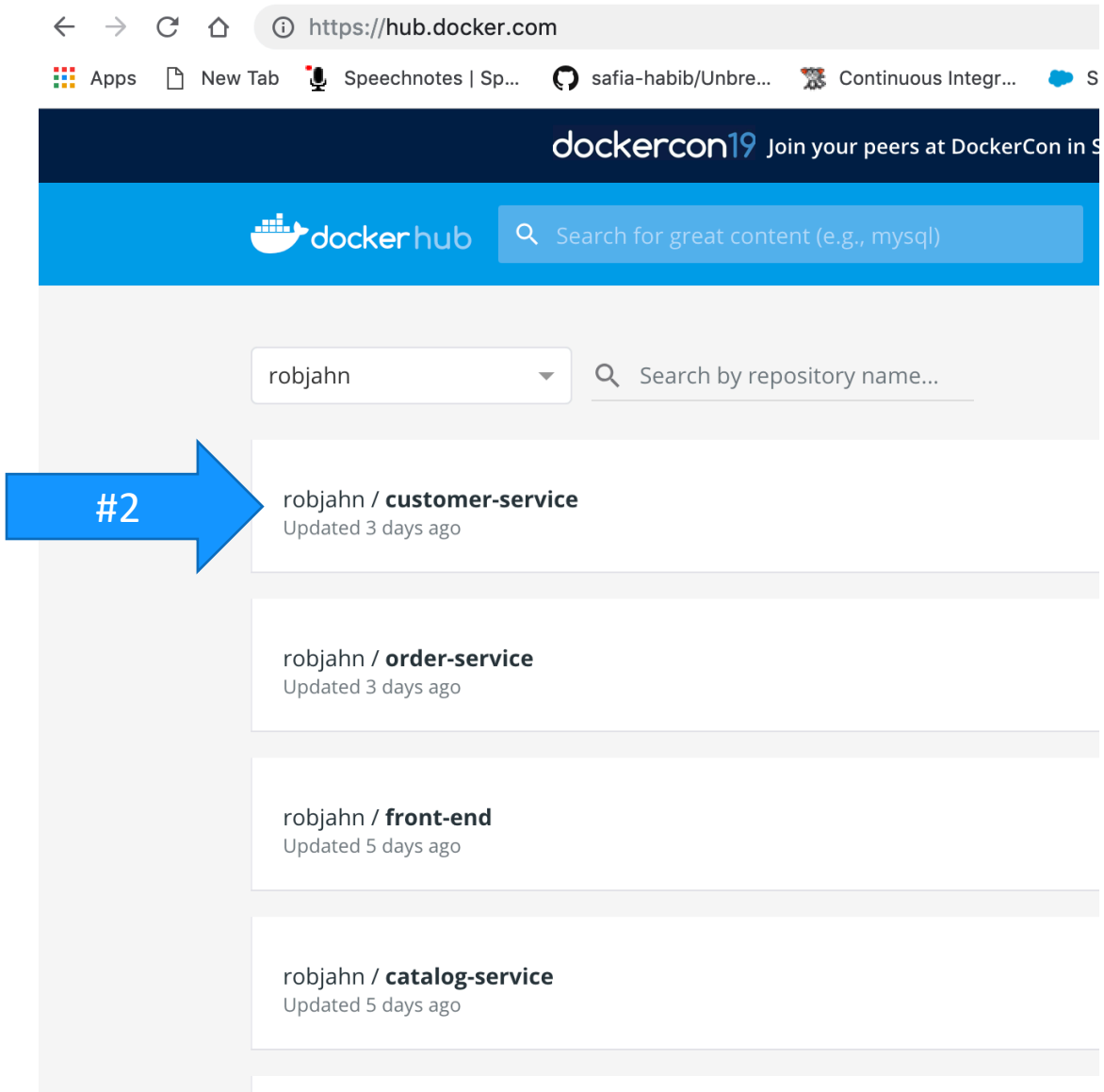
Docker images

1. For the workshop, we pre-built all the services and built docker images. Jenkinsfiles for these builds are in the demo app github services repos
2. Docker images are in Dockerhub
3. The kubernetes deployment files will use this account

#3

```
spec:
  replicas: 1
  selector:
    matchLabels:
      run: customer
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: customer
    spec:
      containers:
      - image: robjahn/customer-service:1
        name: customer
        ports:
        - containerPort: 8080
        resources: {}
        imagePullPolicy: Always
status: {}
```

#2



Demo infrastructure setup procedure

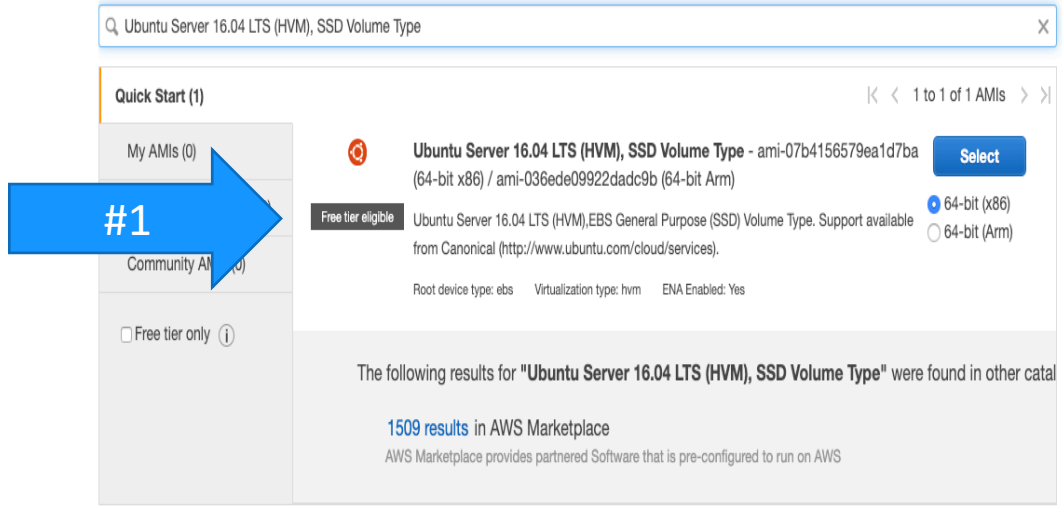
1. Setup EC2 Bastion host
2. Provision Infrastructure: AWS Kubernetes Cluster
3. Generate configuration file using your cheat sheet.
4. Setup Infrastructure: Jenkins, Jenkins Pipelines, Kubernetes workspaces, Dynatrace

Demo infrastructure setup

Setup EC2 Bastion host

Create EC2 instance – One of Two Ways

1. Access github, https://github.com/dt-kube-demo/setup-infra/blob/master/AWS.md
2. Create bastion host
   - With script, this is the approach we will take.
     - https://github.com/pcjeffmac/pipelineWorkshop
   - Manually –
     1. Create New EC2 Instance
        - Search for Ubuntu Server 16.04 LTS (HVM), SSD Volume Type
     2. Create Instance Type
        - Pick type = t2.mocro
     3. Configure Instance Details
        - Leave all defaults
     4. Add Storage
        - Leave all defaults
     5. Add Names
        - Add a tag key = Name. Value = dt-kube-demo-bastion
     6. Configure Security Group
        - Update SSH for TCP 22 to be source = 'My IP'. Pick 'Anywhere'
     7. Launch

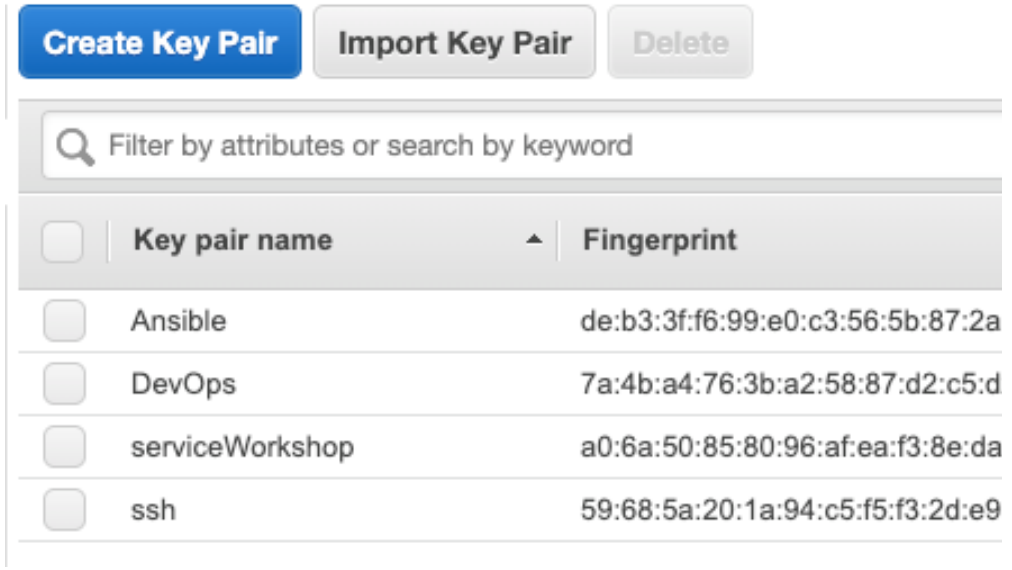


Create AWS bastion via a script

- Assumes you have AWS CLI configured locally
- Configure aws cli
  - Run this command to configure aws CLI
    - aws configure
      - enter AWS Key ID
      - enter AWS Secret Access Key ID
      - enter Default region (us-east-1)
      - enter format (json)
- Validate aws cli access
  - Run this command
    - aws iam list-access-keys

Create Bastion via aws cli

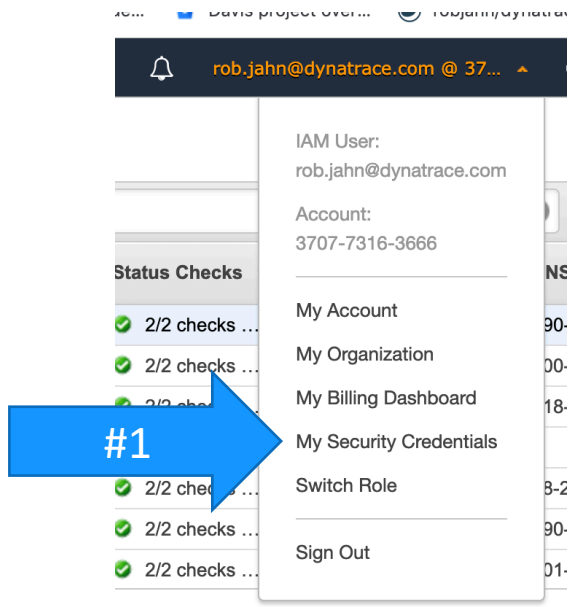
- Create EC2 Key Pair
  - EC2->Key Pair->Create Key Pair
- Download script
  - https://github.com/pcjeffmac/pipelineWorkshop
  - bastion.sh
- Modify Script
  - SSH_KEY=<your ssh aws key name>
    - The script will now create this key for you.
  - CLUSTER_REGION=<example us-east-1>
  - RESOURCE_PREFIX=<example your last name>
- Run script
  - ./bastion.sh

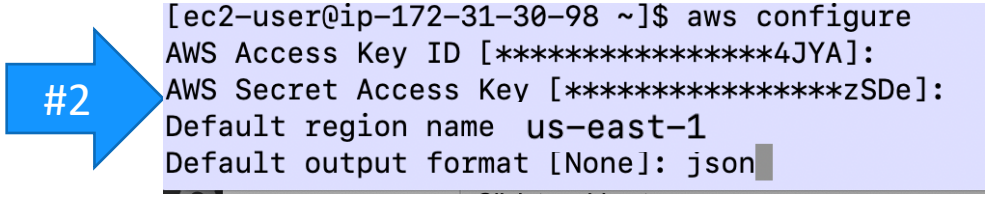


AWS CLI configuration

After you are connected to the bastion host….

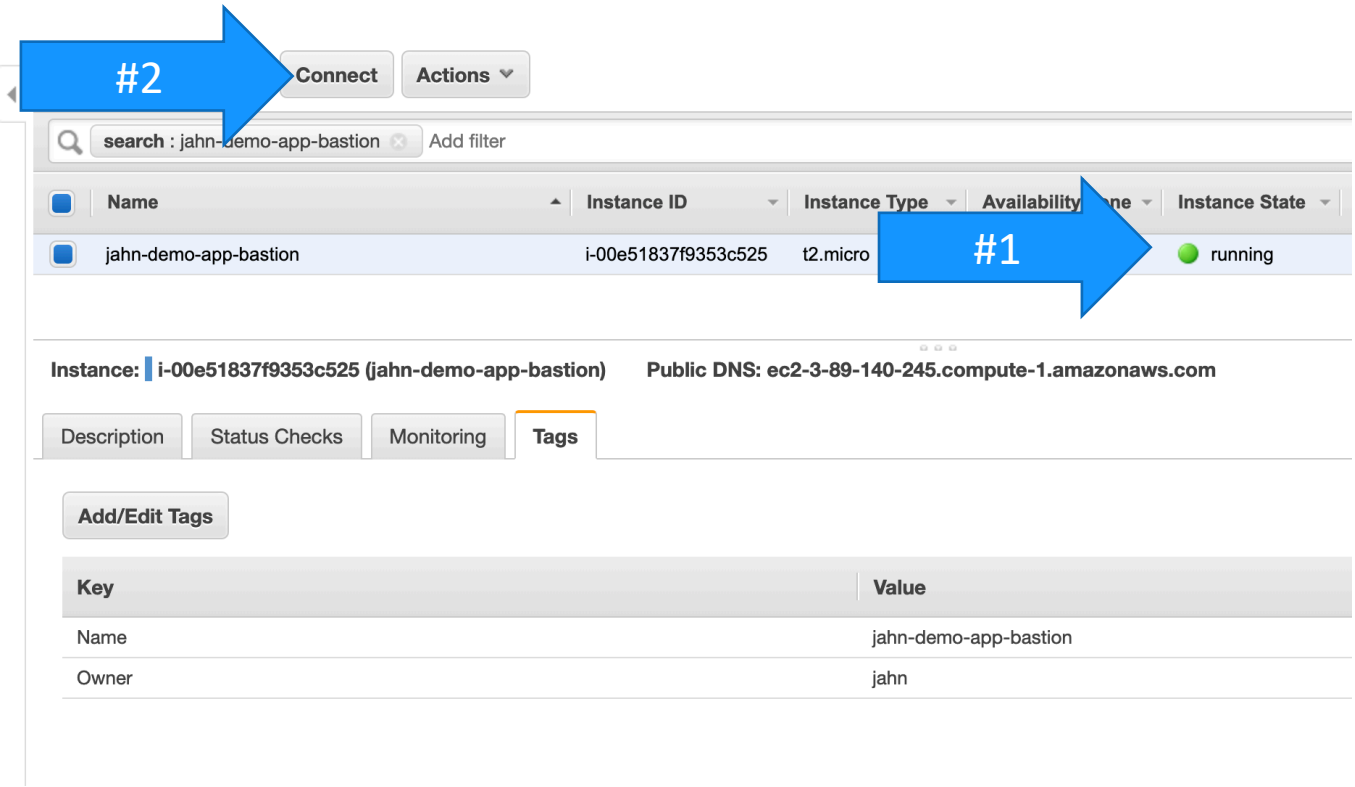
1. You should have saved your AWS Access Key, AWS secret Access key. If not, go back to my security credentials and make a new key
   - https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html
2. To configure AWS cli configure command, run these commands:
   - aws configure
     - Enter your AWS Access Key, AWS secret Access key and default region name. choose 'us-east-1'. Pick JSON as the output format
     - NOTE: if you use something other that 'us-east-1' then you will have the adjust the terraform variables for region in the terraform AWS cluster provisioning script.
3. Validate connection by running some aws commands
   - aws ec2 describe-instances
     - This should show json out of the ec2 instances as a way to confirm connectivity





Connect to EC2 instance

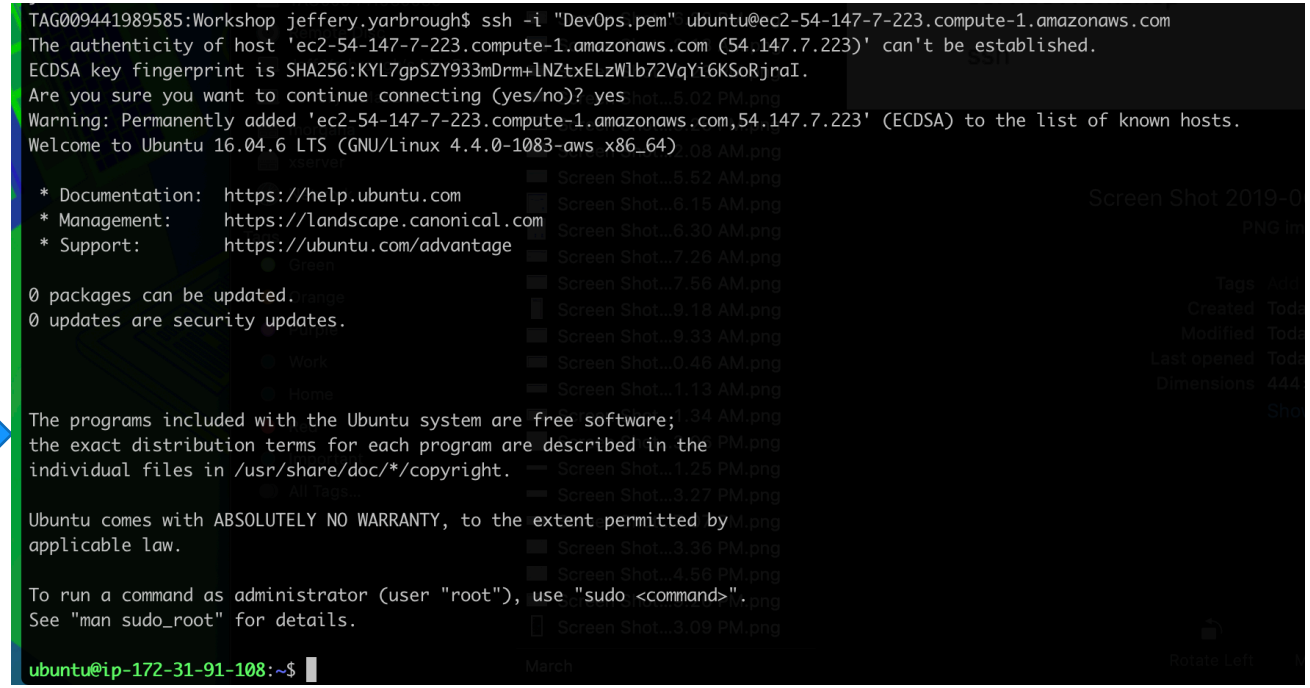
1. Go back to the EC2 summary page and wait until it is running
2. Then check the row of the instance and click the ‘connect’ button



Connect to EC2 instance

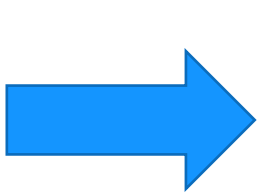
1. Connect to host. For example in Mac @ the terminal in folder with pem file
   - ssh -i ”lastname.pem" ubuntu@ec2-18-232-185-178.compute-1.amazonaws.com
2. If using windows you can use Putty & Puttygen to make your pem file a ppk file https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html?icmpid=docs_ec2_console
3. You should be logging now

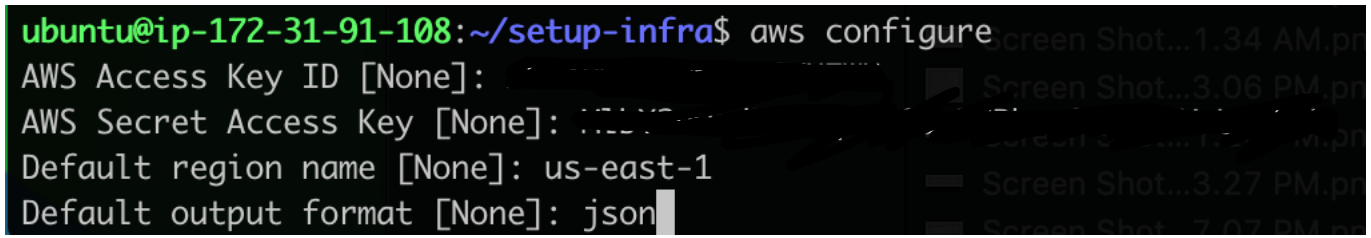
#1



Initialize aws CLI on the bastion

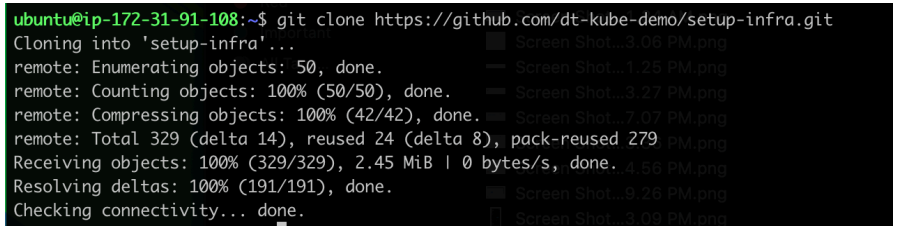
1. Run these commands
   - sudo apt update
   - sudo apt install awscli –yes
   - aws configure
     - enter AWS Key ID
     - enter AWS Secret Access Key ID
     - enter Default region (us-east-1)
     - enter format (json)
   - aws ec2 describe-instances

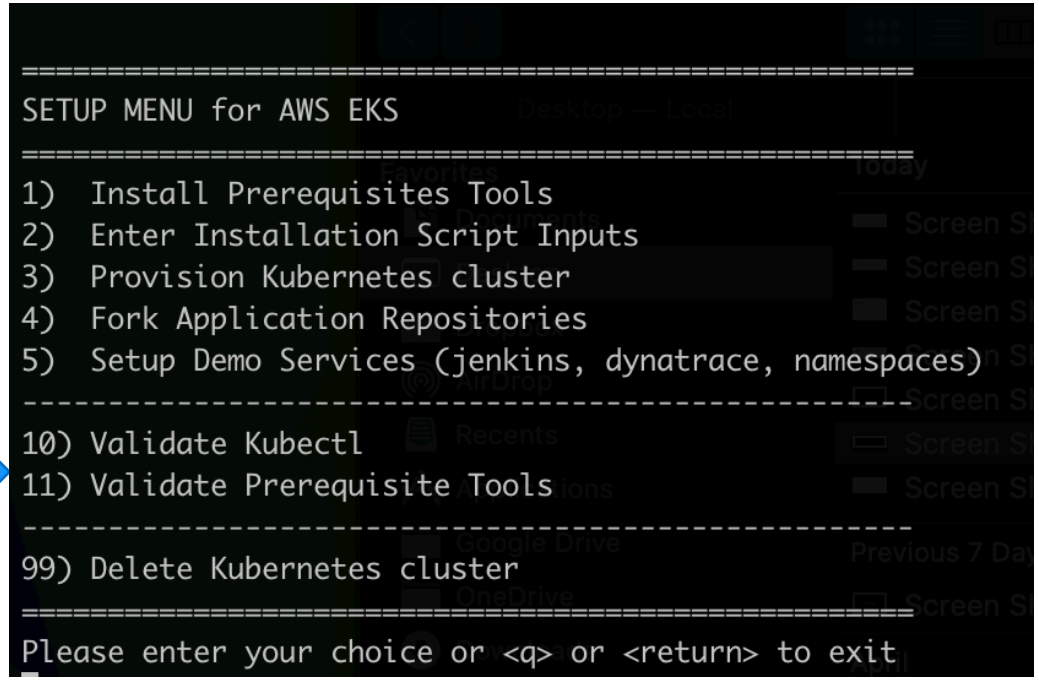




Install Prerequisits Tools

1. Clone GitHub repo
   - git clone https://github.com/dt-kube-demo/setup-infra.git
   - cd setup-infra
2. This will install unix tools such as kubctl, jq, etc…
   - Run
     - ./setup.sh <deployment type> (./setup.sh eks)
       - eks = AWS
       - aks = Azure
   - Select Option 1





Demo infrastructure setup

Provision Infrastructure

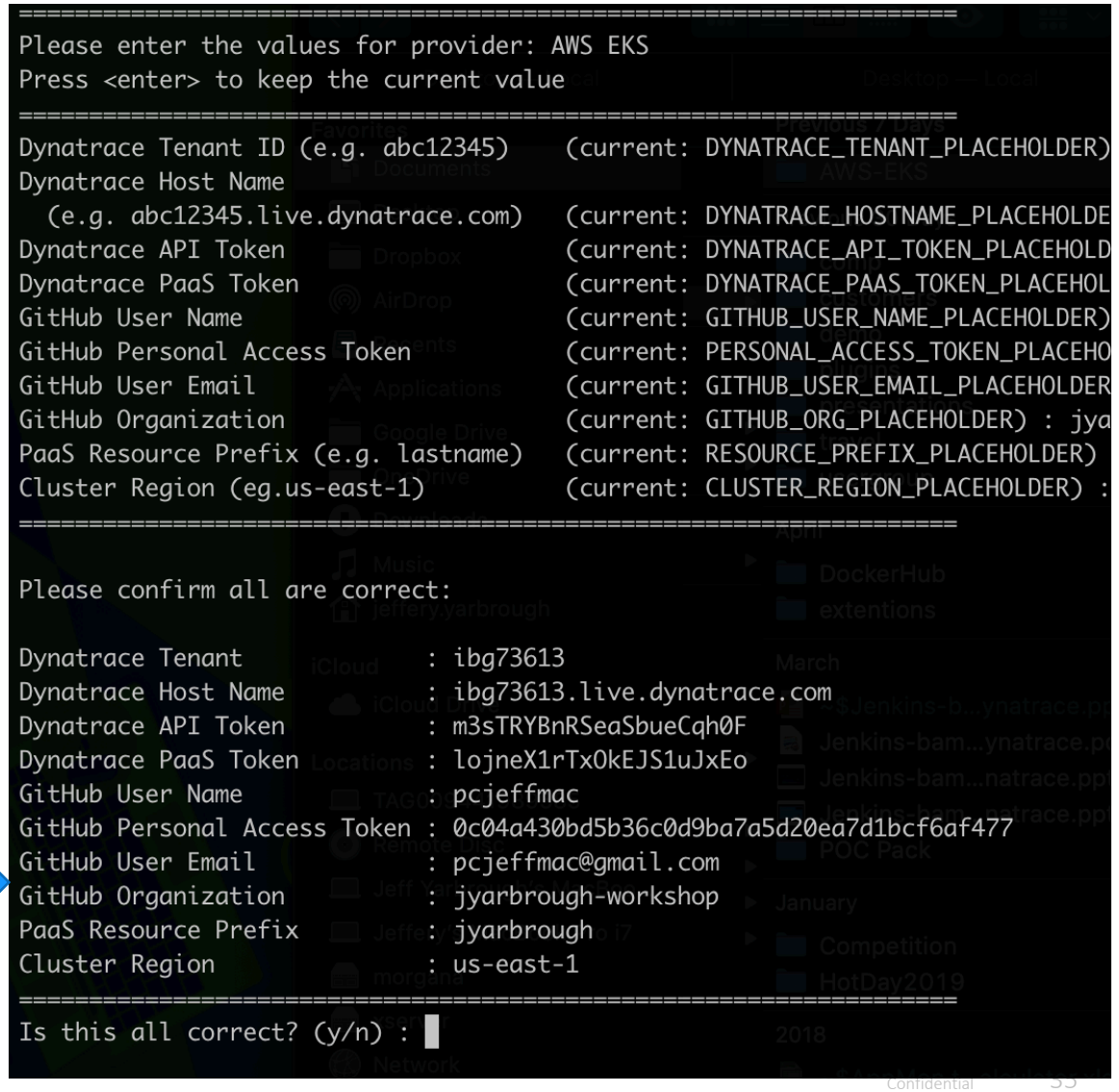
Prepare bastion host – Enter Installation Script Inputs

Back on the bastion host, to fork the demo app repos, run these commands:

- cd ~/setup-infra/
- ./setup.sh eks

This script will prompt for values and save it into a file called ‘creds.json’

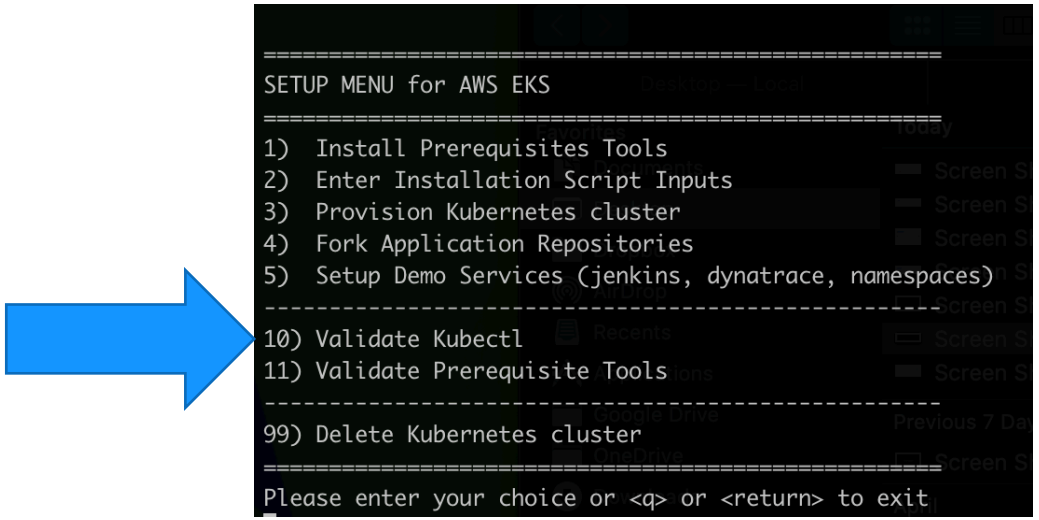
Then select ‘y’



Prepare bastion host – Provision Kubernetes Cluster

Back on the bastion host, to fork the demo app repos, run these commands:

- cd ~/setup-infra/
- ./setup.sh eks

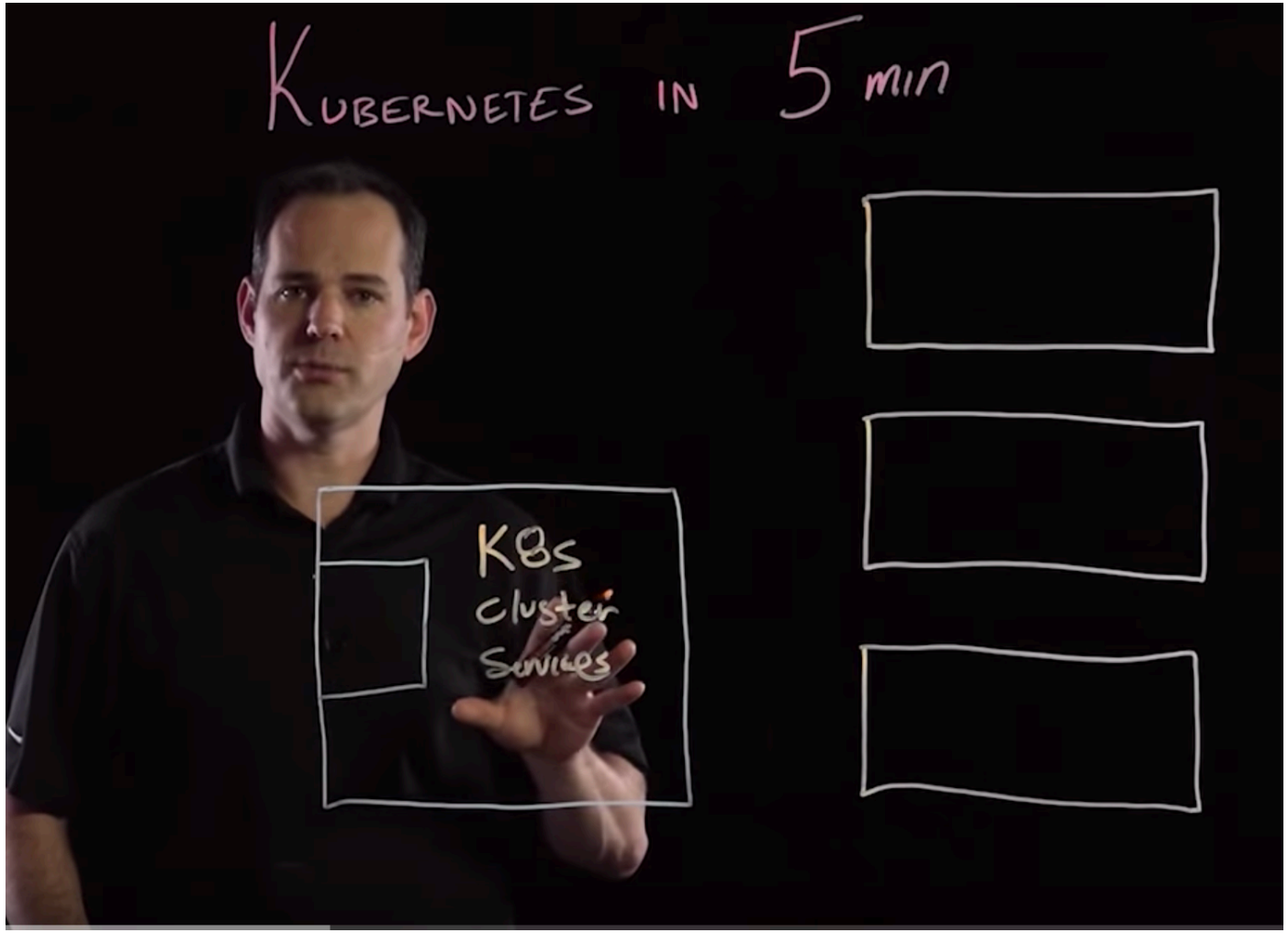


Select Option 3 (should take ~10-15 min.)

Provision the Kubernetes Cluster

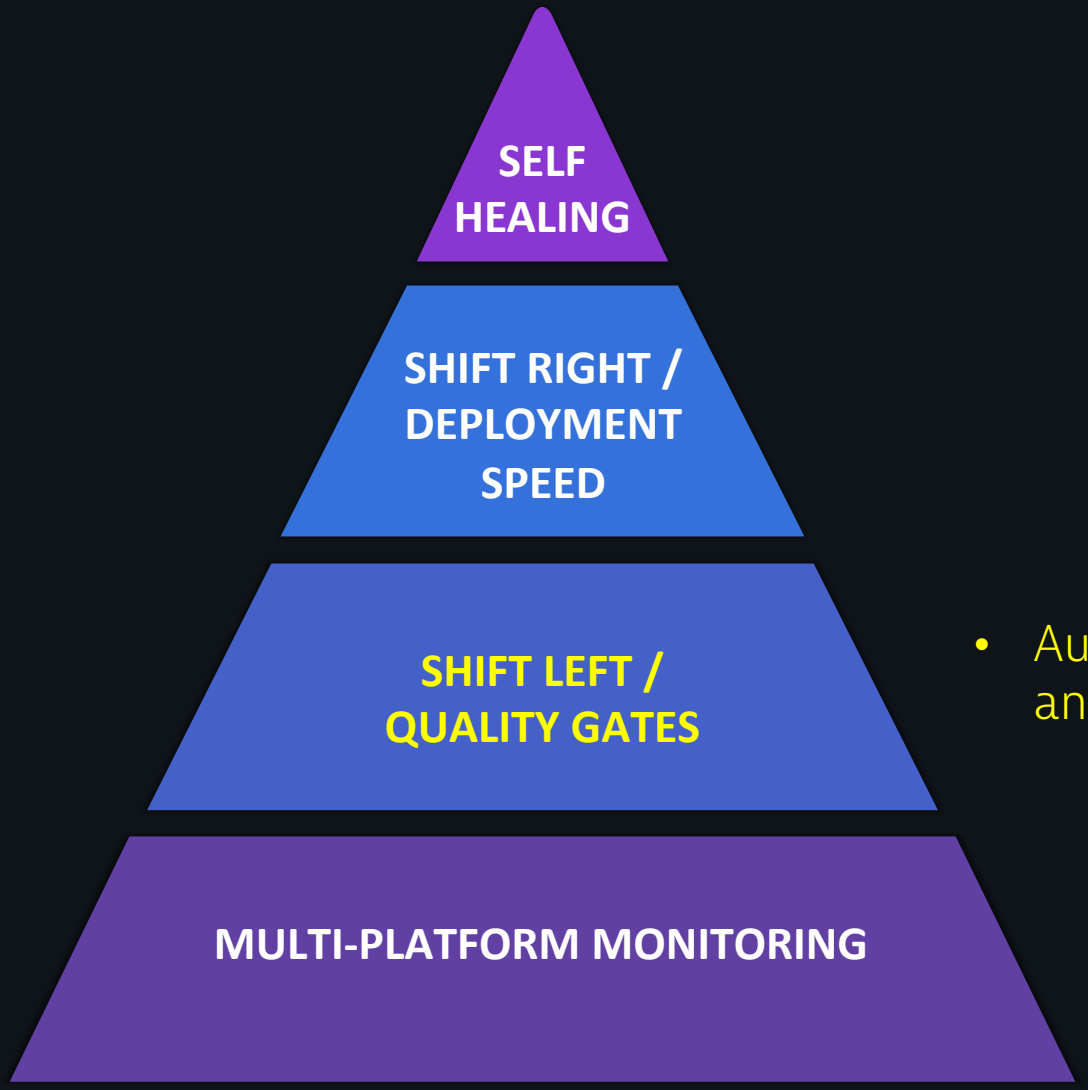
While the cluster is building…

Kubernetes



https://www.youtube.com/watch?v=PH-2FfFD2PU

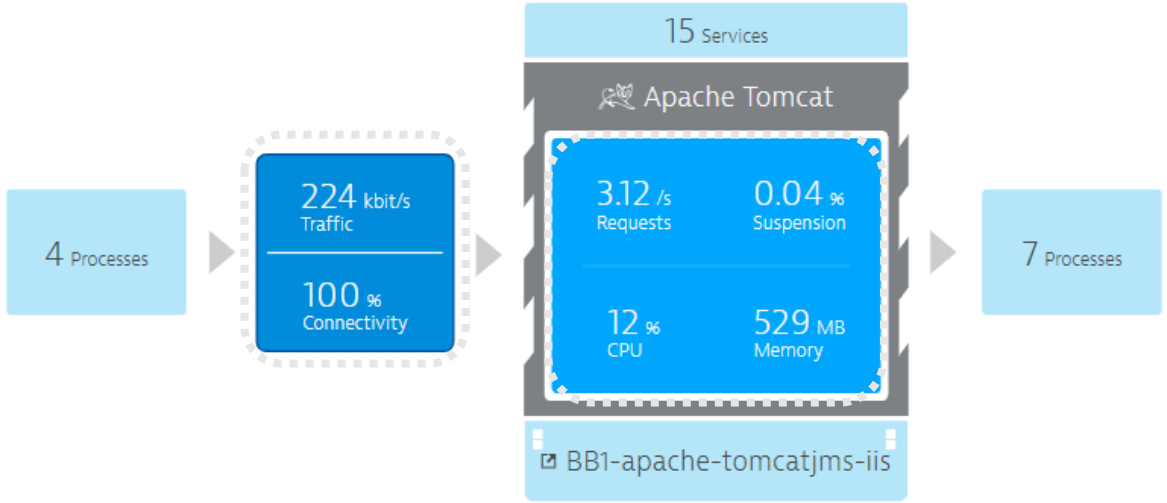
Key blue print – unbreakable continuous delivery model



- Automate quality (shift-left) – automate the pipeline and stop bad code changes before they reach prod

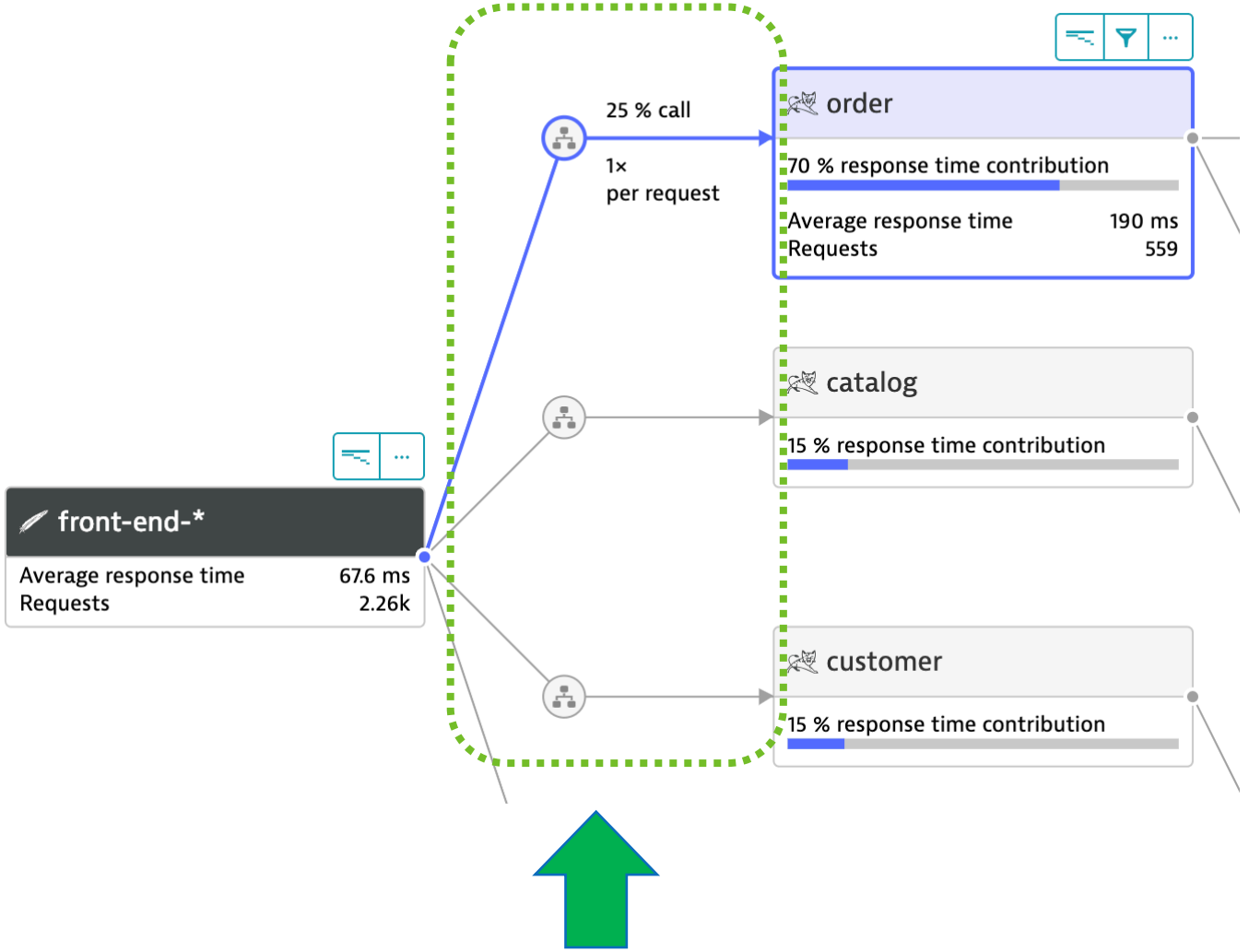
Quality Gate: Detect Change in Resource Consumption

- Use Cases
  - Bad coding leads to higher costs?
- Metrics
  - Memory Usage
  - Bytes Sent/Received
  - Overall CPU
  - CPU per Transaction Type
- How to query?
  - Some through CloudWatch API
  - Dynatrace Timeseries API



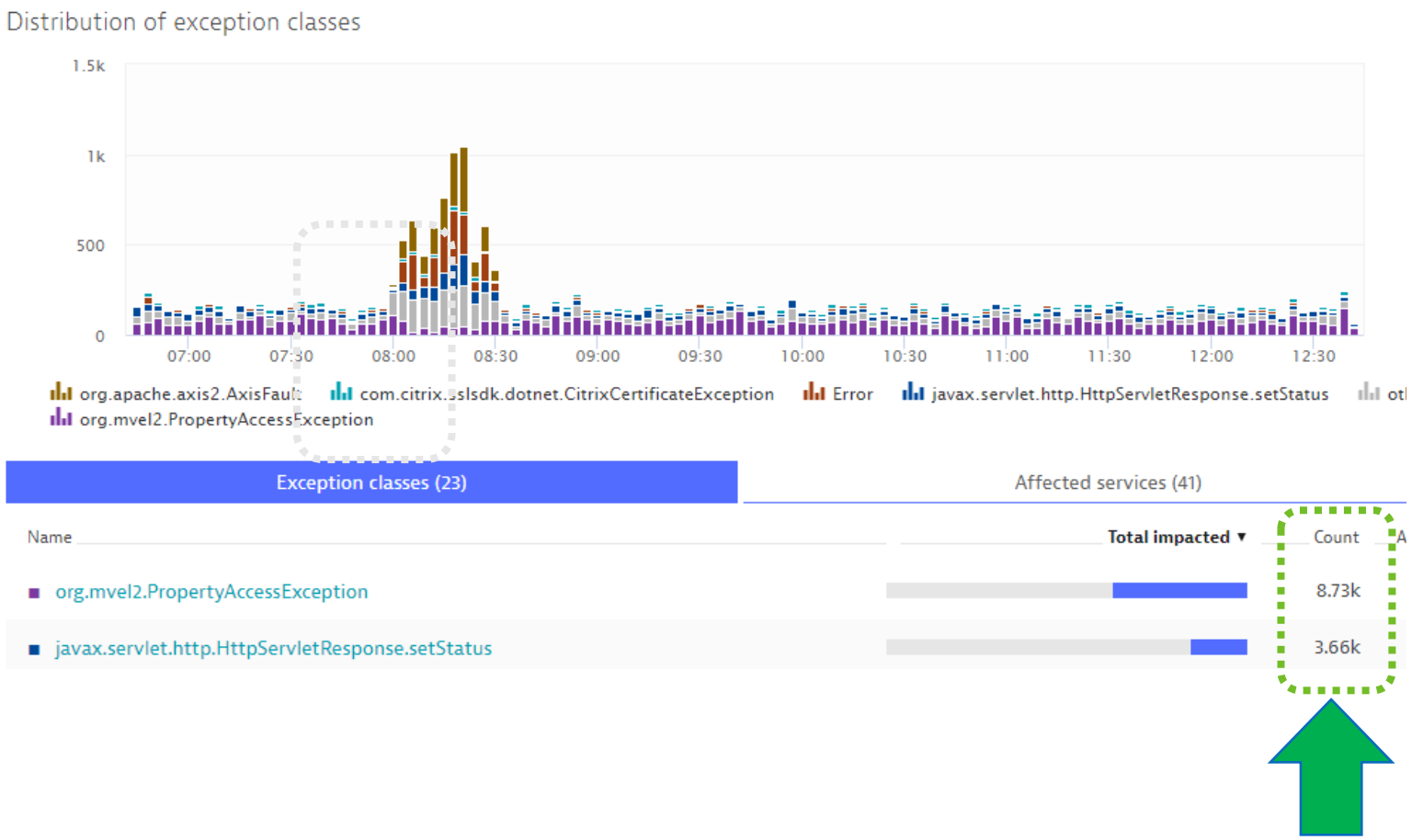
Quality Gate: Detect Change in Dependencies

- Use Cases
  - Do we have new dependencies? On Purpose?
  - Are we connecting to the services we are supposed to connect?
  - How many container instances are required?
- Metrics
  - Number of incoming / outgoing dependencies
  - Number of instances running on
- How to query?
  - Maybe CloudWatch API
  - Dynatrace SmartScape API

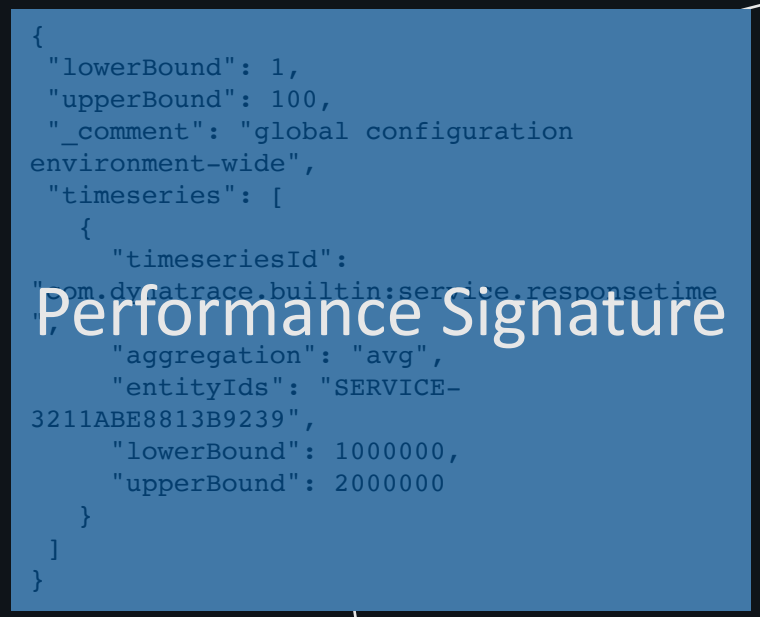


Quality Gate: Detect Change in Application Exception Handling

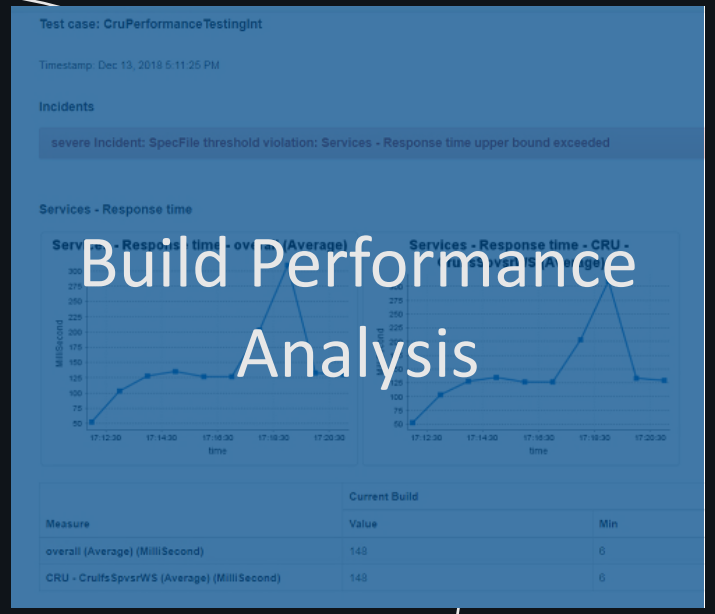
- Use Cases
  - Did we introduce new “hidden” exceptions?
- Metrics
  - Total Exceptions
  - Exceptions by Class&Service
- How to query?
  - Dynatrace Timeseries API



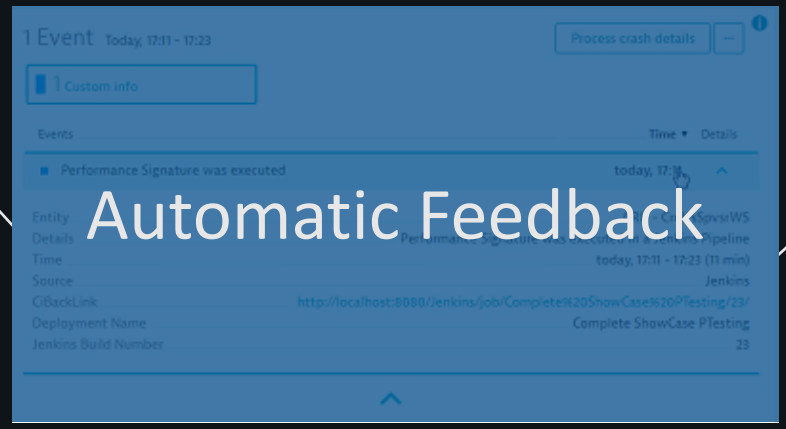
Shift-left quality gates – monitoring-spec-as-code



Performance Signature



Build Performance Analysis



Automatic Feedback

Integrate performance signature – monitoring-as-code

1 Code / Config Change

2 Monitoring Spec

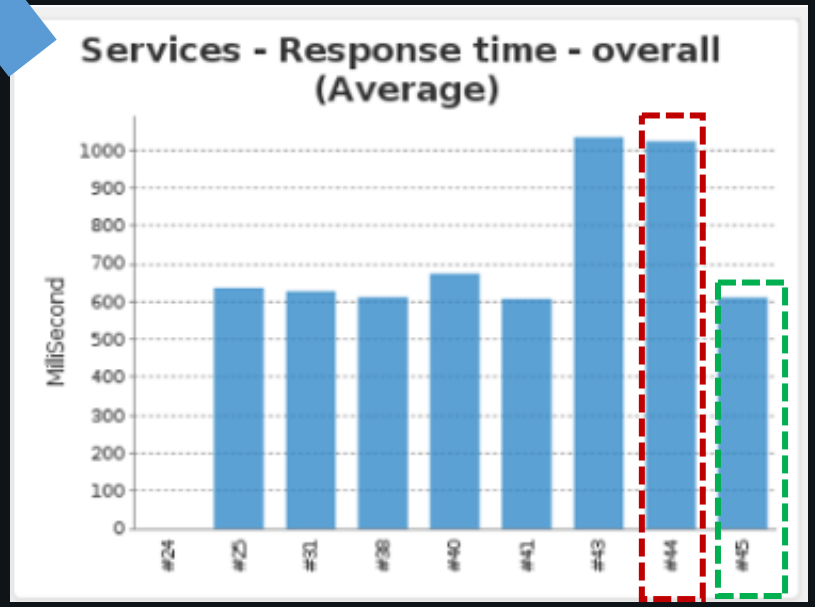
3 Staging

4 Monitoring

5 End Users

YES / NO ?

```
{
 "lowerBound": 100,
 "upperBound": 1000,
 "_comment": "global configuration environment-wide",
 "timeseries": [
   {
     "timeseriesId": "service.responsetime",
     "aggregation": "avg",
     "entityIds": "SERVICE-3211ABE8813B9239",
     "lowerBound": 20,
     "upperBound": 300
   }
 ]
}
```



Shift-left quality gates – performance-diagnostic-as-self-serve

Top database statements

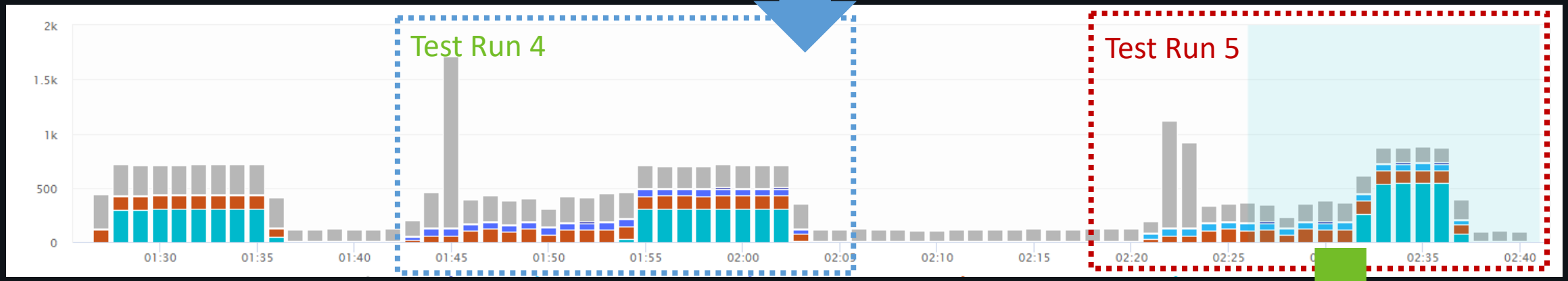
Analyze the most frequent and most expensive database statements in monitored server-side applications.

Top web requests

Understand and analyze which web requests are the most expensive and most frequently called.

Exception analysis

Understand and analyze all code-level exceptions in monitored server-side applications.



Other important quality gates & metrics

Check 1
- Bad coding leading to higher costs?

- ❑ **Metrics**
- ❑ Memory usage
- ❑ Bytes sent / received
- ❑ Overall CPU
- ❑ CPU per transaction type

Check 2
- New dependencies? On Purpose?
- Services connecting accurately?
- Number of container instances needed?

- ❑ **Metrics**
- ❑ Number of incoming / outgoing dependencies
- ❑ Number of instances running on containers

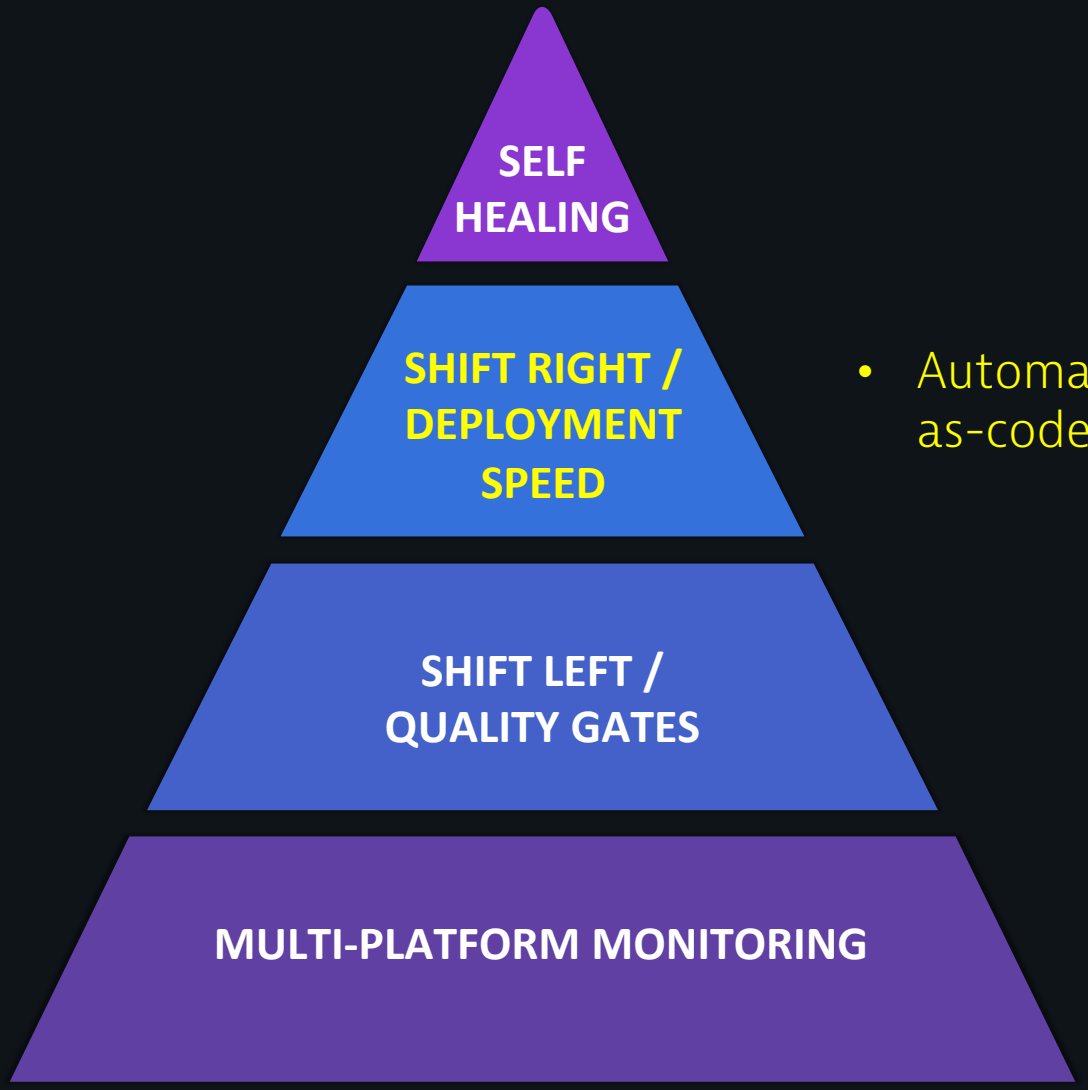
Check 3
- Are we jeopardizing our SLAs?
- Does load balancing work?
- Difference between Canaries?

- ❑ **Metrics**
- ❑ Response Time (Percentiles)
- ❑ Throughput & Per per Instance / Canary

Check 4
- Did we introduce new "hidden" exceptions?

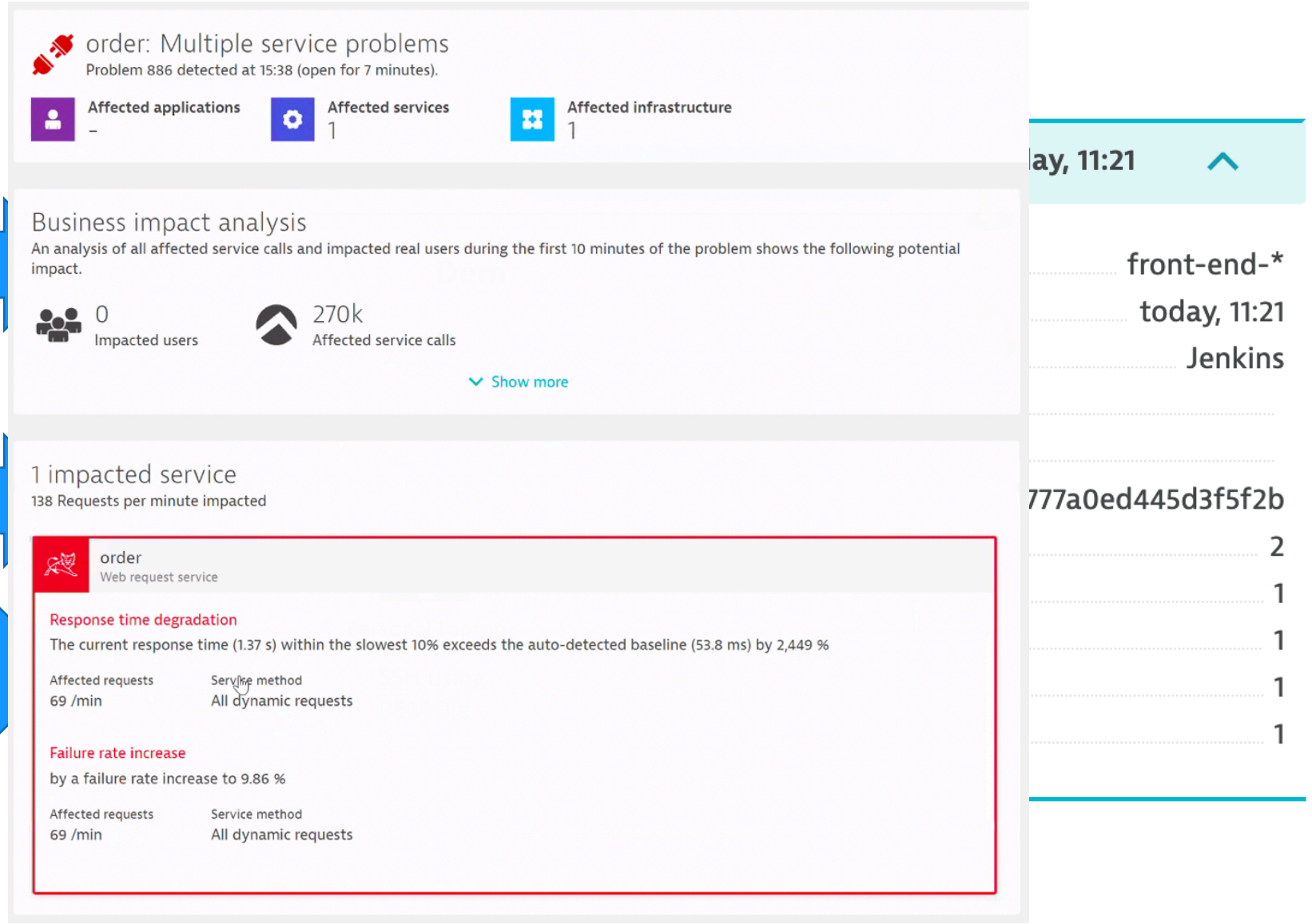
- ❑ **Metrics**
- ❑ Total Exceptions
- ❑ Exceptions by Class & Service

Key blue print – unbreakable continuous delivery model

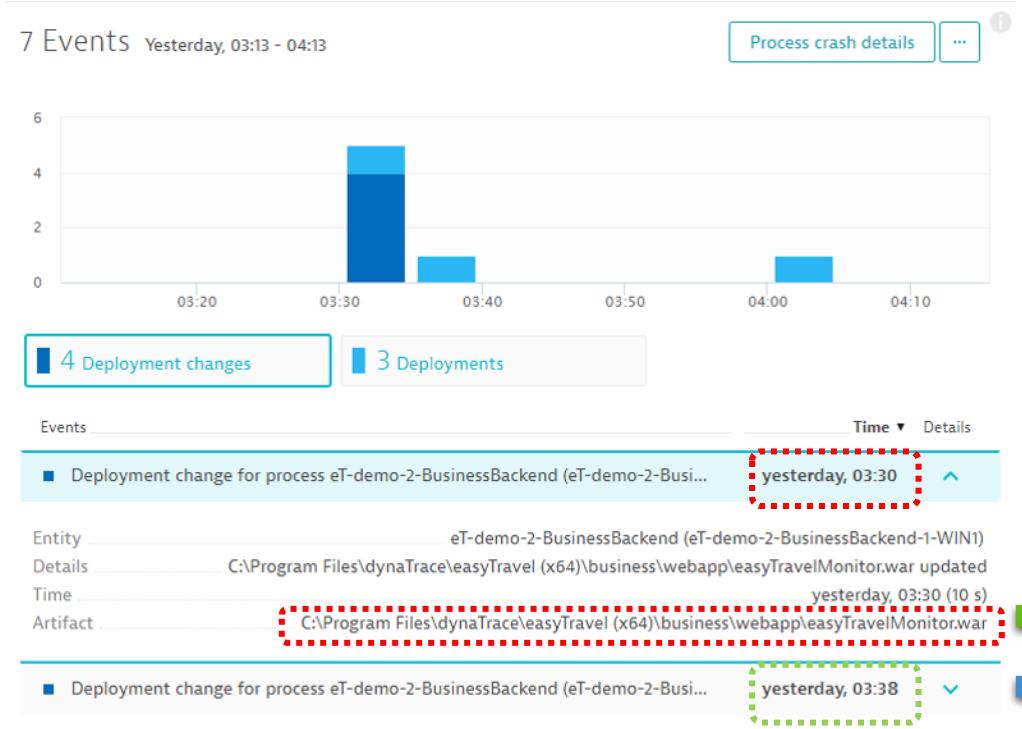


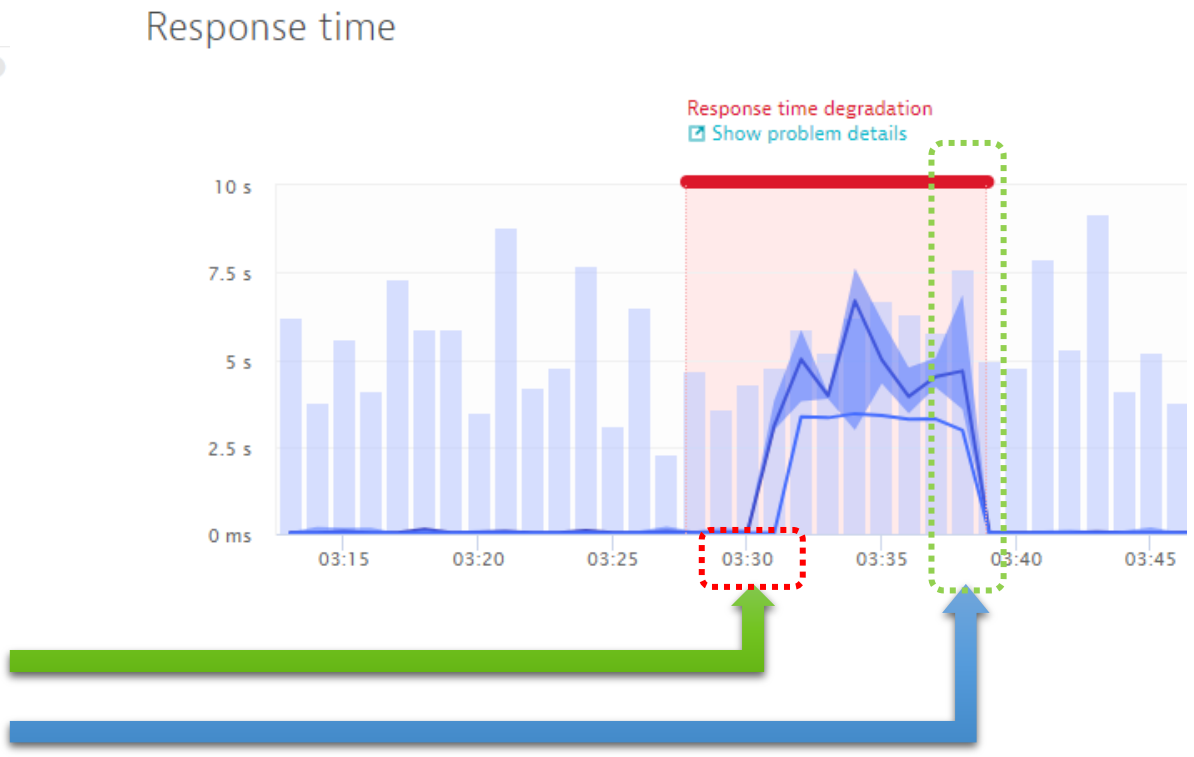
- Automate deployment (shift-right) – push “monitoring-as-code” for auto-validation and auto-alerting

Context of Code/Config changes during problem

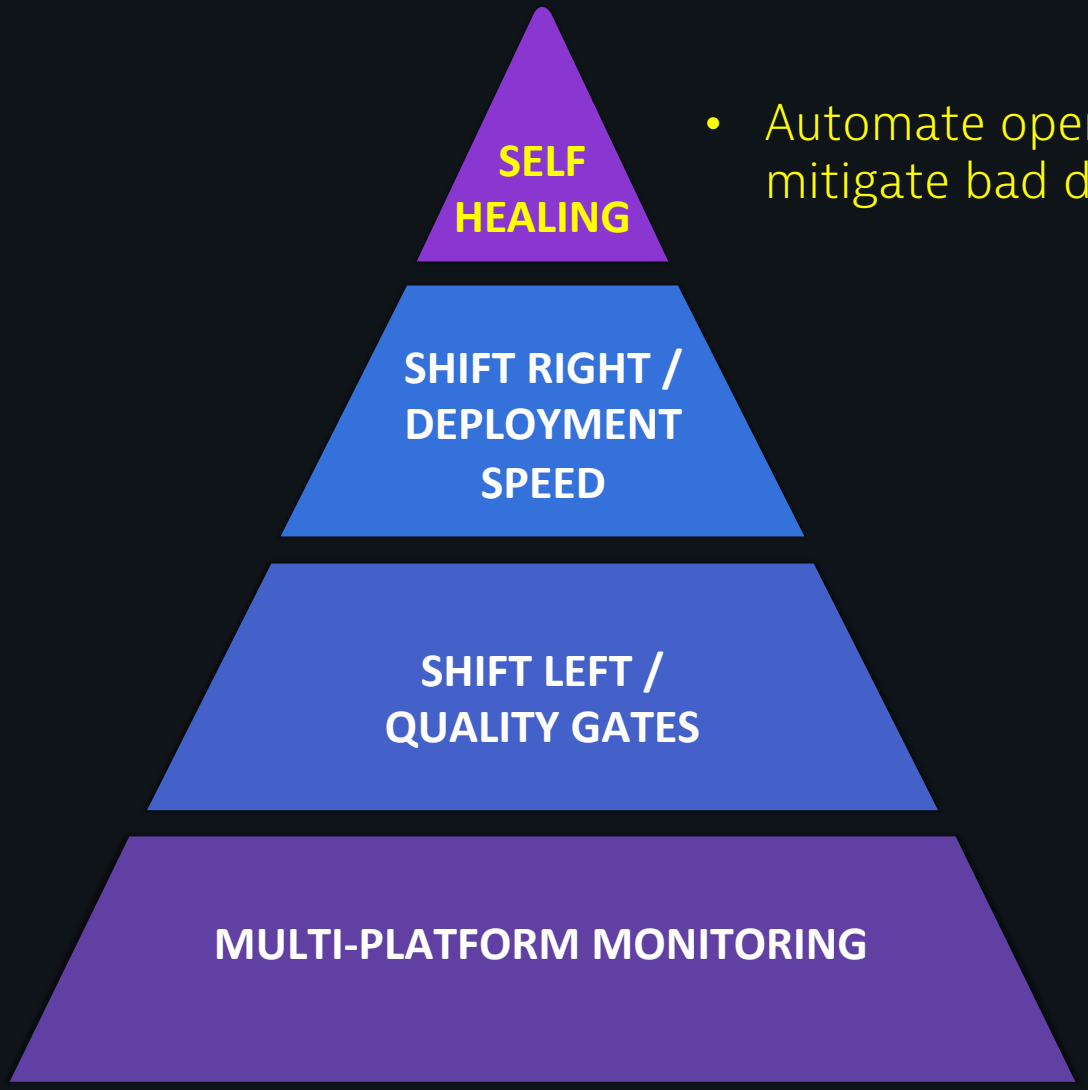


Provides more context to your environment



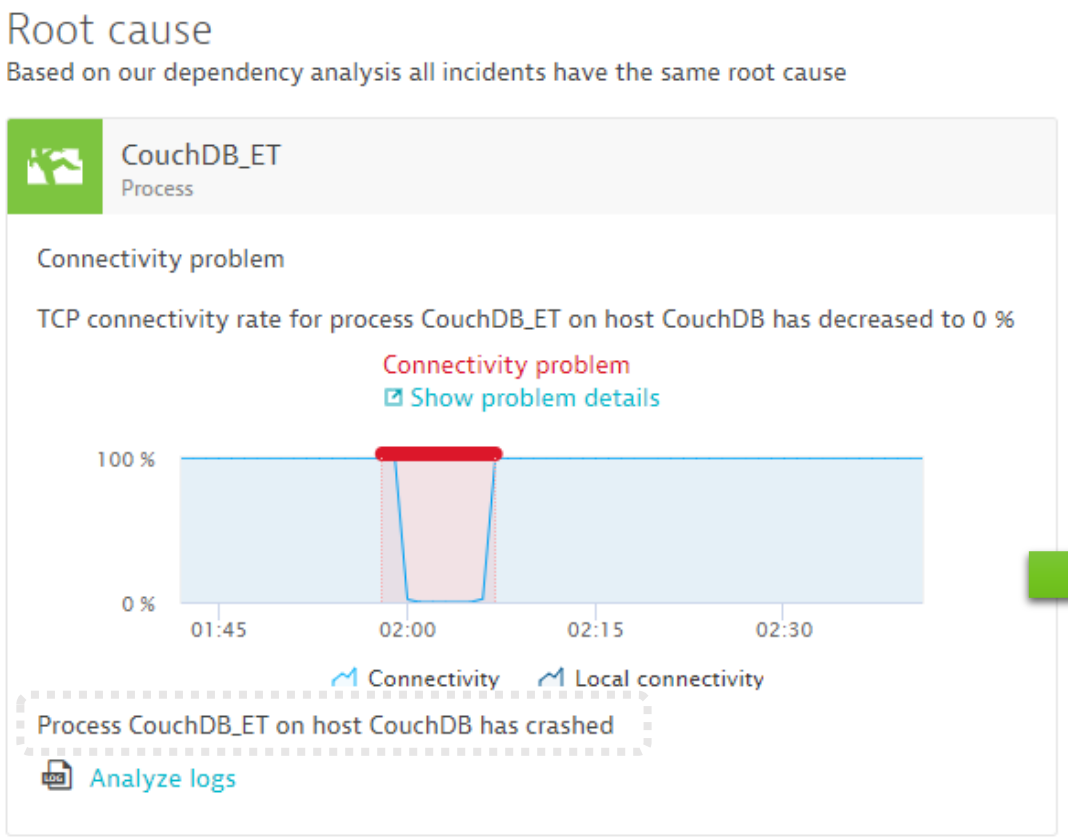


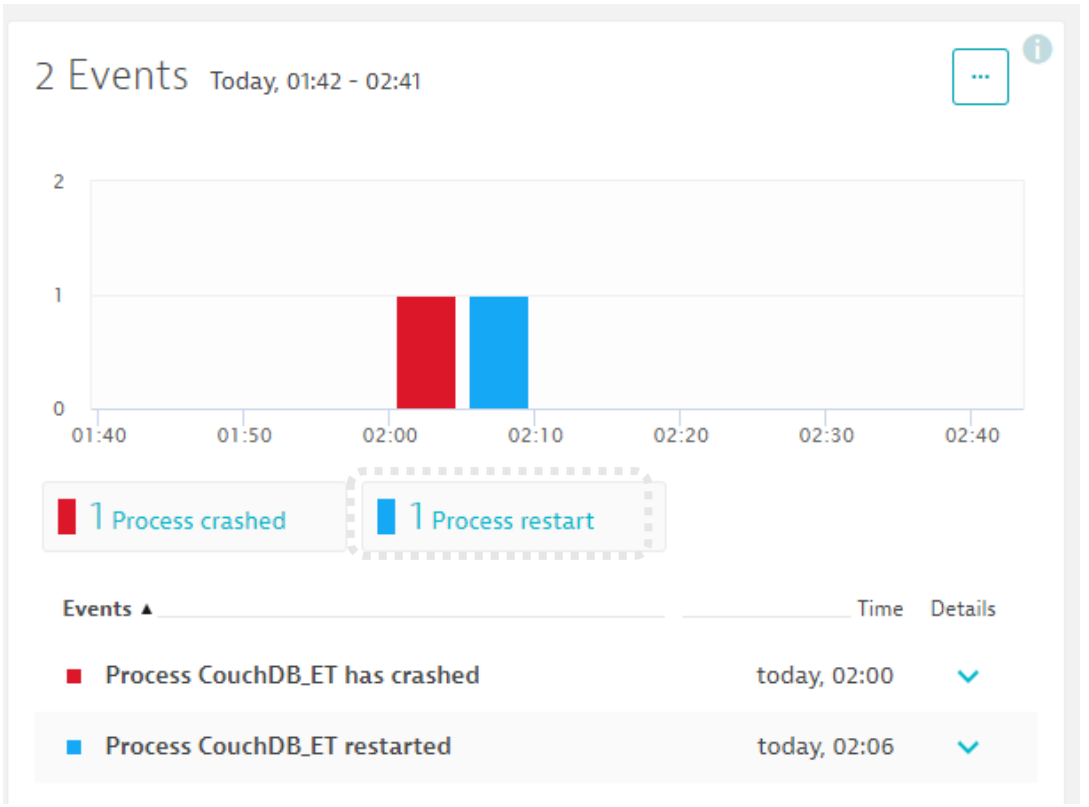
Key blue print – unbreakable continuous delivery model



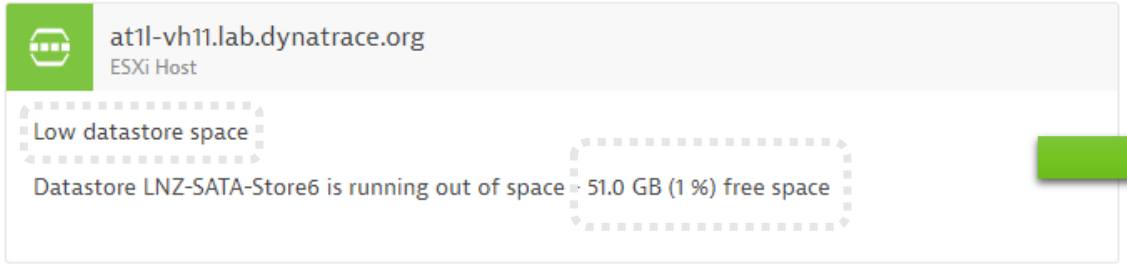
- Automate operations (self-healing) – auto-mitigate bad deployments in production

Use Case: Crash -> Restart

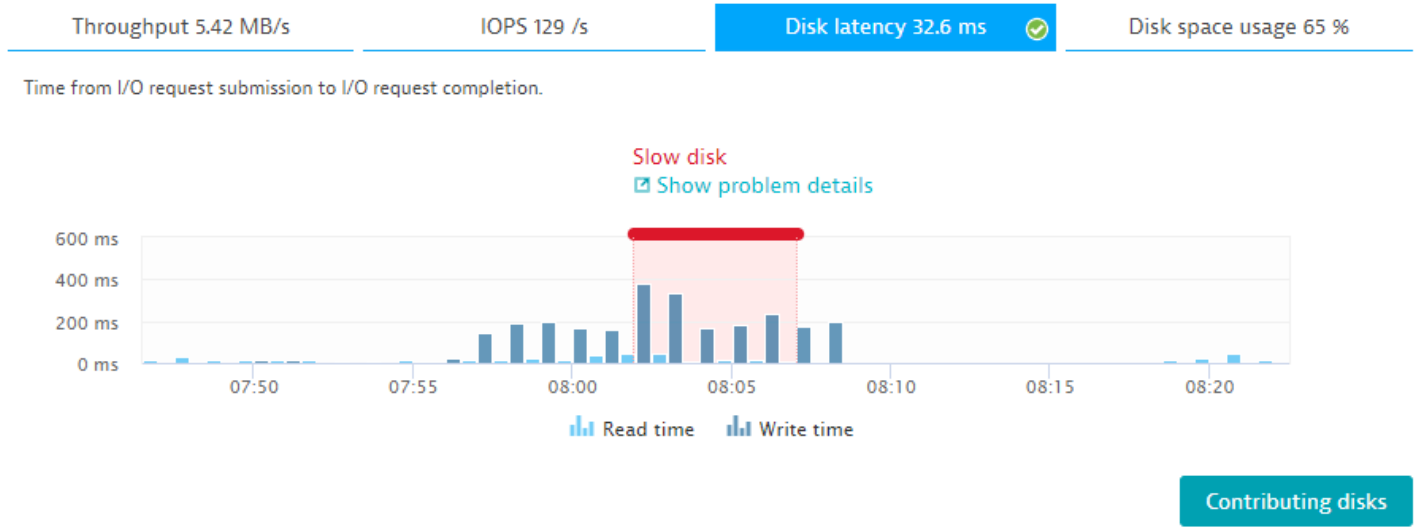




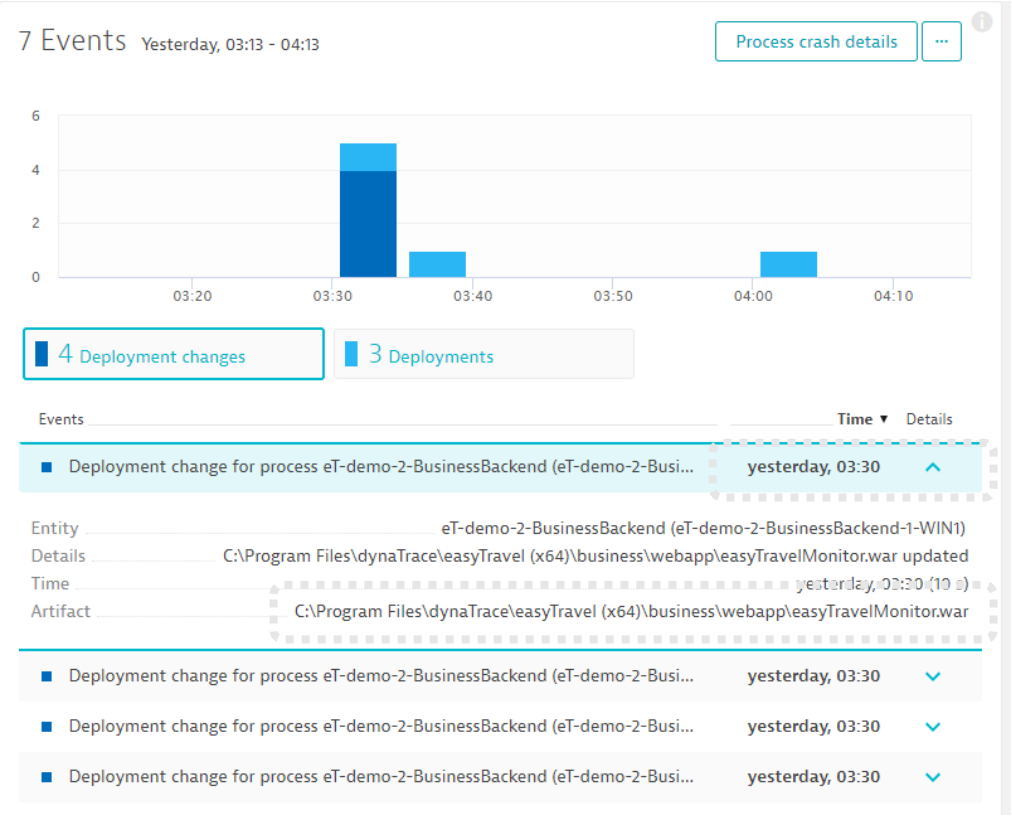
Use Case: Full or slow Disk -> Clean Up

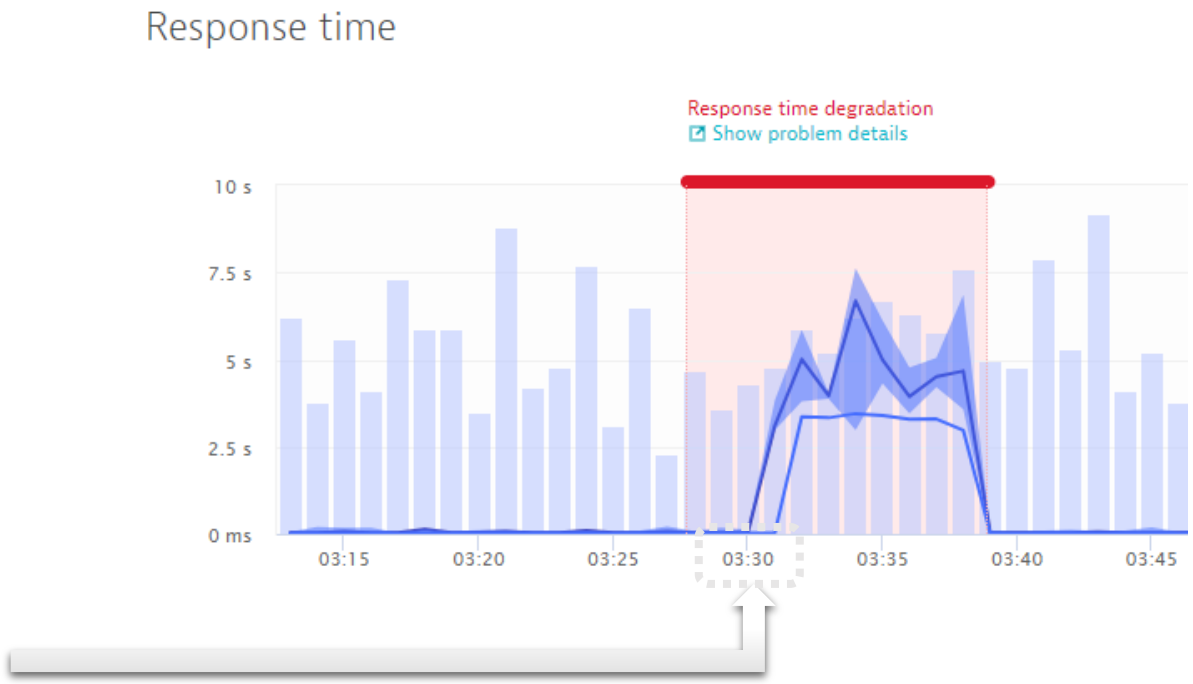


$ find ./my_dir -mtime +10 -type f -delete



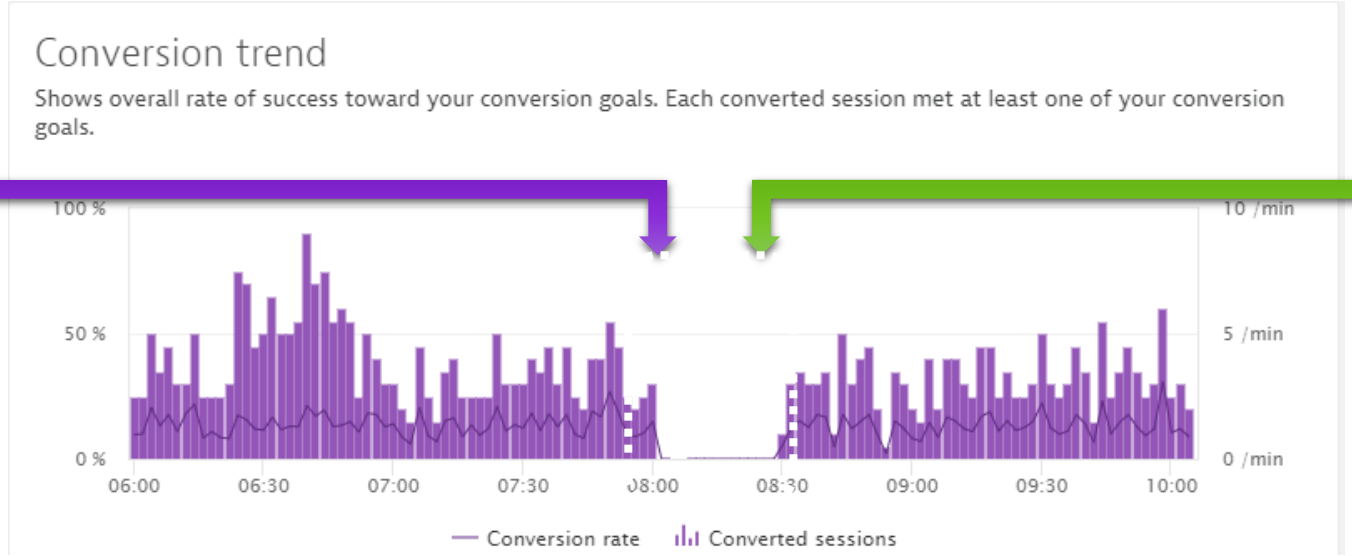
Use Case: Bad Configuration Changes -> Revert





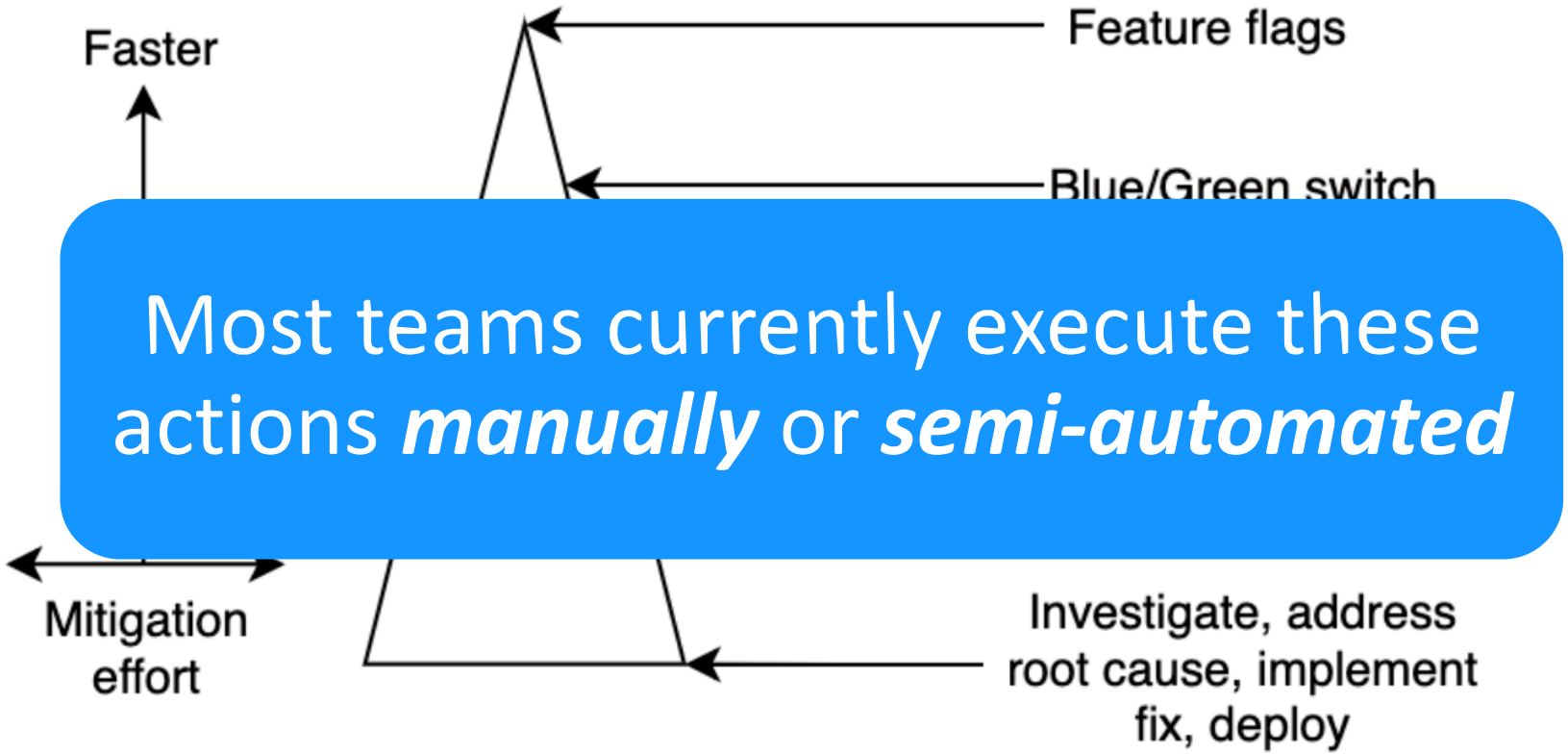
Use Case: End User Impact -> Reverse Blue / Green

Deploy Blue



Back to Green

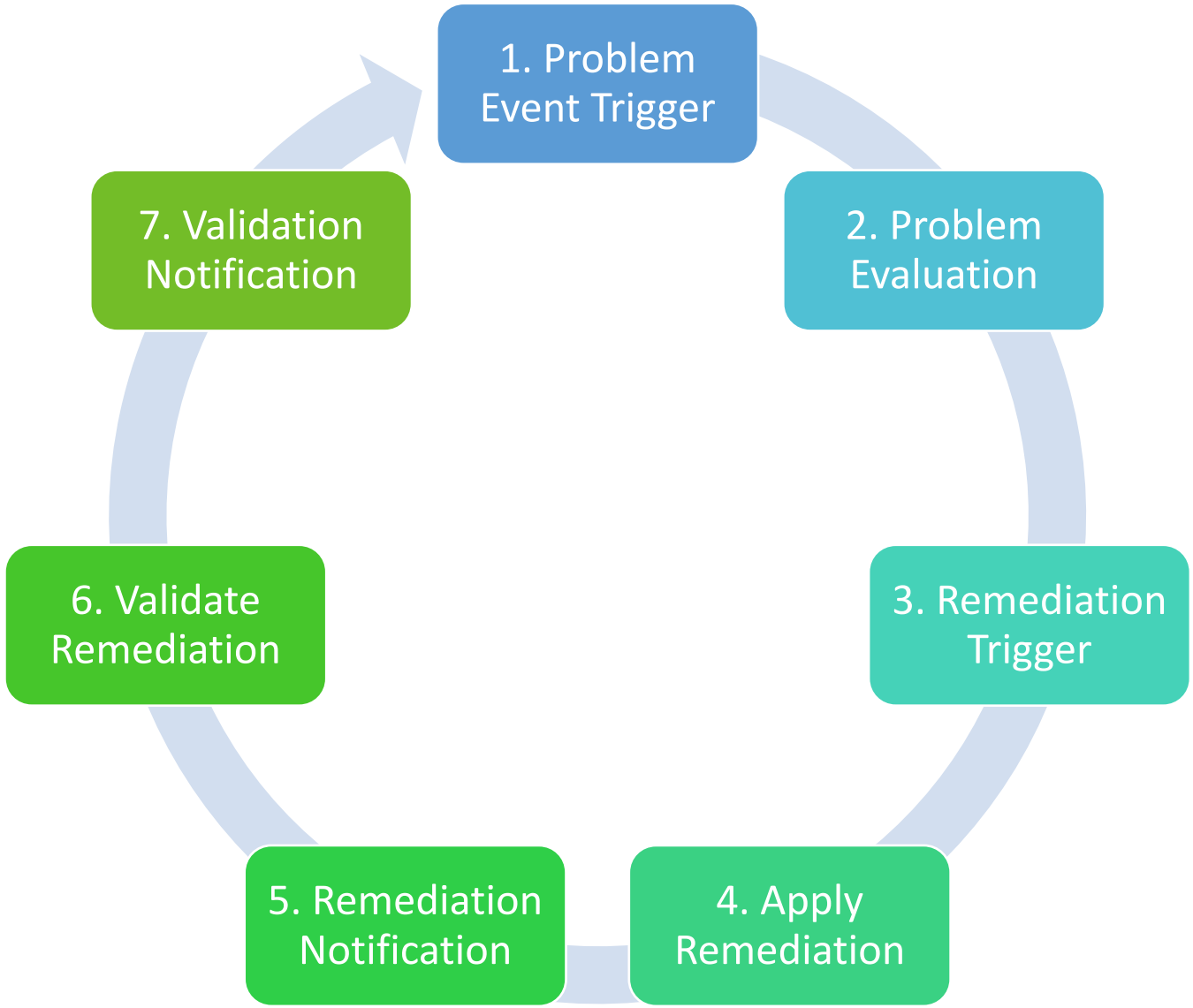
Time to take action ….



Most teams currently execute these actions ***manually*** or ***semi-automated***

https://medium.com/@sashman90/ops-mitigation-triangle-300c81d97df6

Auto Remediation Closed Loop System



1 User Impact Problem Evolution

2 Self-healing Automation

Application / Cloud Infrastructure Monitoring Tool

2:00 a.m. escalate?

Auto-mitigate!

```
 "lowerBound": 100,
 "upperBound": 1000,
 "_comment": "global configuration
environment-wide",
 "timeseries": [
   {
     "timeseriesId":
"service.responsetime",
     "aggregation": "avg",
     "entityIds": "SERVICE-
3211ABE8813B9239",
     "lowerBound": 20,
     "upperBound": 300
   }
 ]
}
```

Add new metric to quality gate – adding automation here, too!

1. CPU Exhausted? Add a new service instance!
2. High Garbage Collection? Adjust/Revert Memory Settings!
3. Issue with BLUE only? Switch back to GREEN!
4. Hung threads? Restart Service!

? Impact Mitigated? → Update Dev Tickets

5 Ongoing? Roll-back → Mark Bad Commits

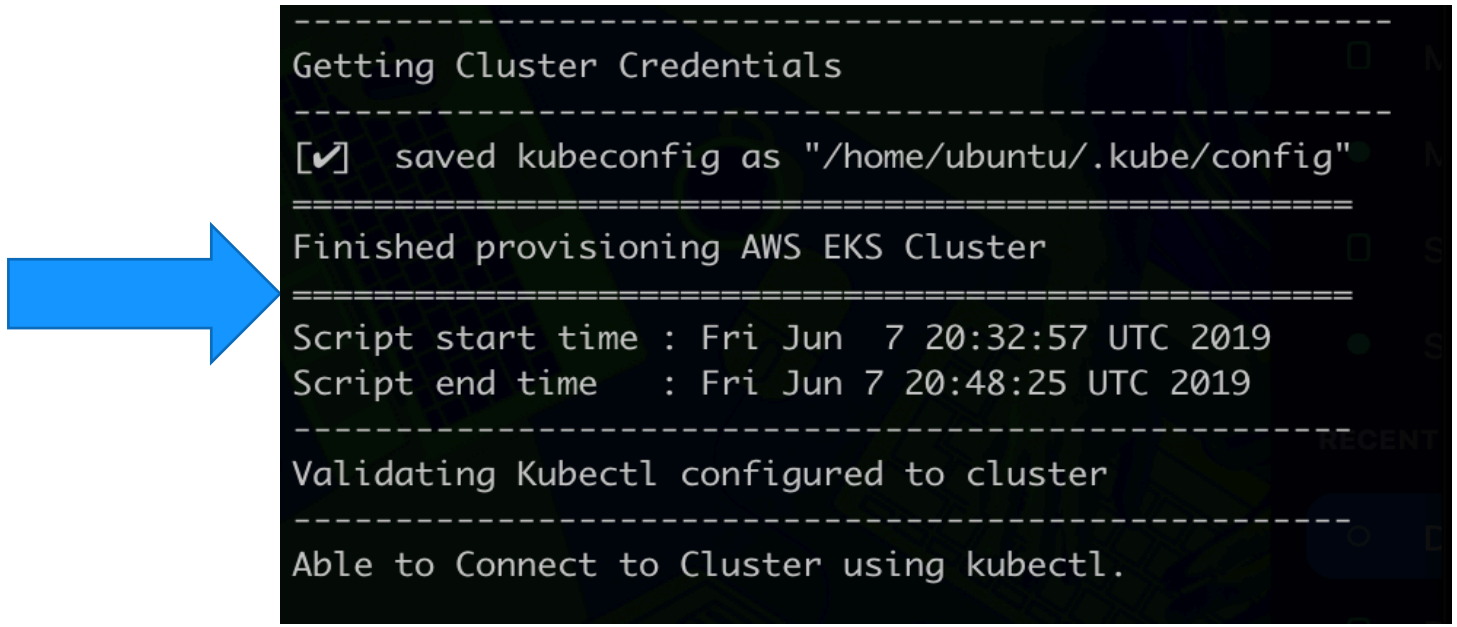
? Still ongoing? → Escalate

Demo infrastructure setup

Complete Infrastructure Setup

Run Setup Provision Cluster, continued

1. You should see a successful validation message at the end

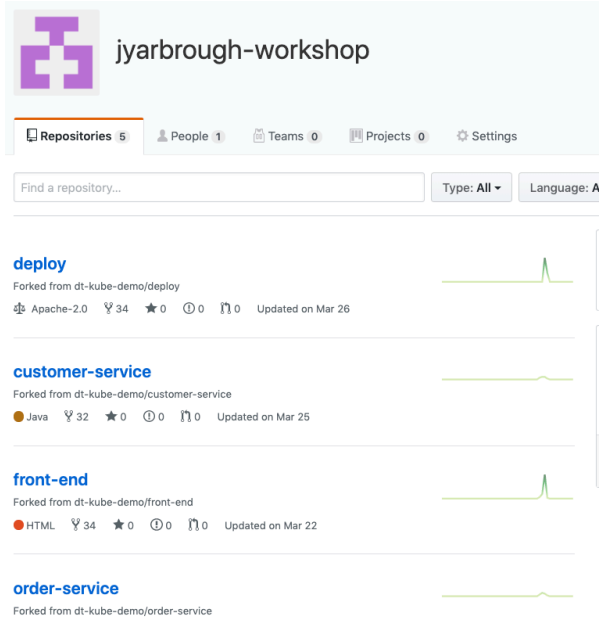


Prepare bastion host – Fork repo

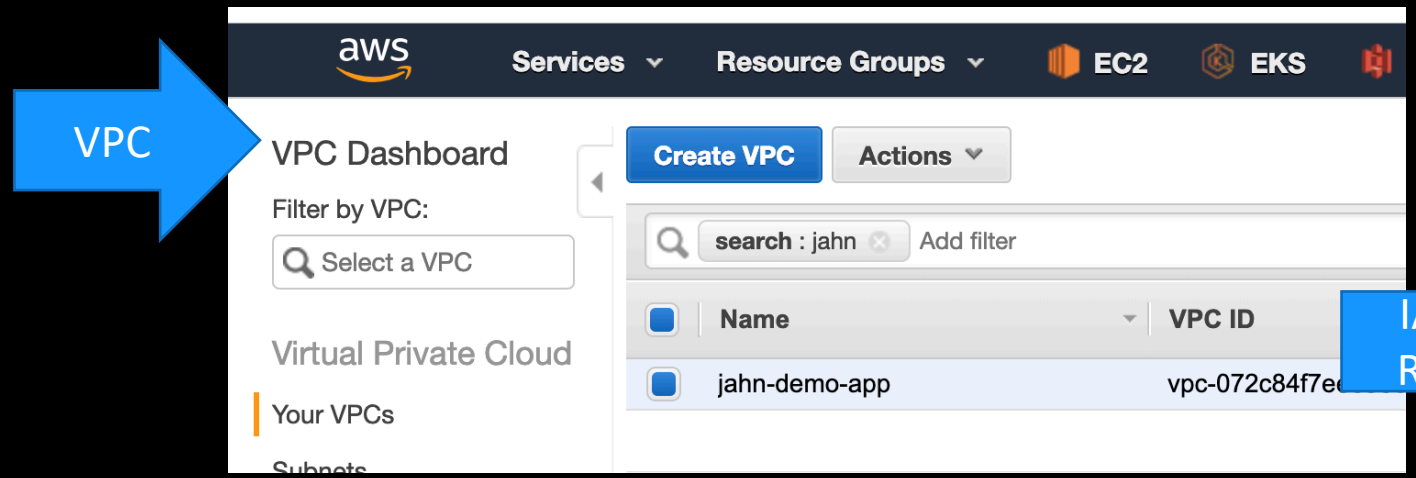
1. Back on the bastion host, to fork the demo app repos,
   - ./setup.sh eks
2. Select 4) Fork Application Repository
   - github.com username:
   - gituHub PW:
3. In git hub, view new Org to verify new repos

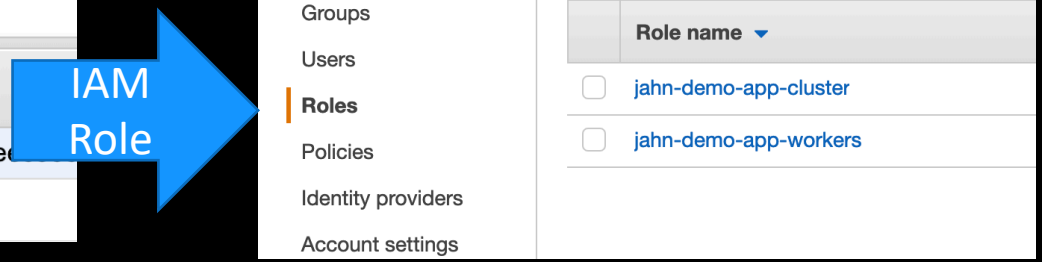
```
================================================
SETUP MENU for AWS EKS
================================================
1)  Install Prerequisites Tools
2)  Enter Installation Script Inputs
3)  Provision Kubernetes cluster
4)  Fork Application Repositories
5)  Setup Demo Services (jenkins, dynatrace, namespaces)
------------------------------------------------
10) Validate Kubectl
11) Validate Prerequisite Tools
------------------------------------------------
99) Delete Kubernetes cluster
================================================
Please enter your choice or <q> or <return> to exit
```

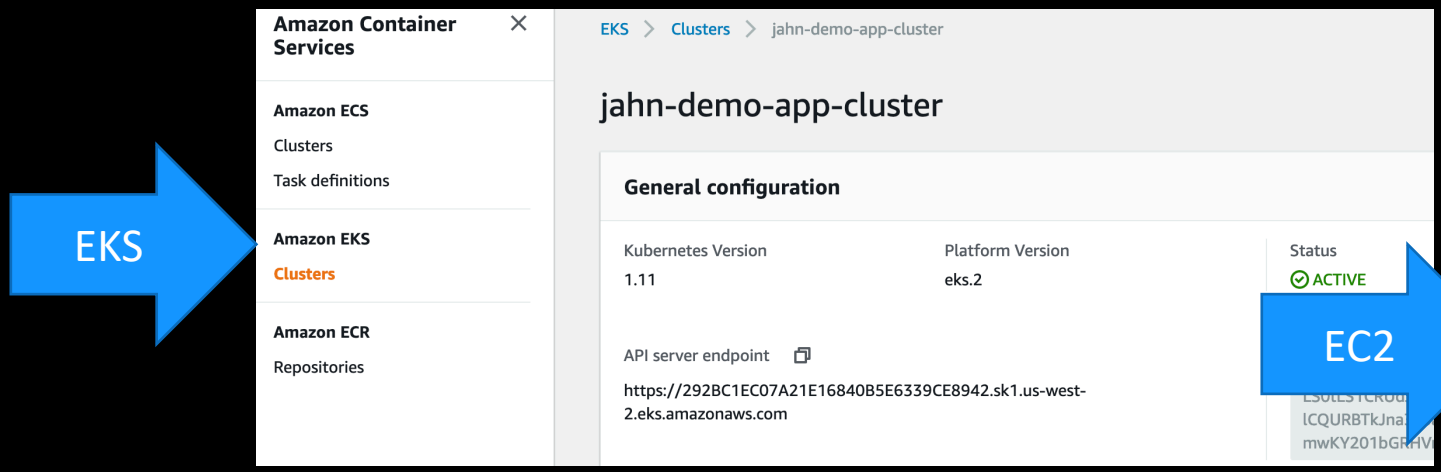
```
================================================
Cloning https://github.com/dt-kube-demo/catalog-service
Forking catalog-service to jyarbrough-workshop
github.com username: pcjeffmac
github.com password for pcjeffmac (never stored):
Updating jyarbrough-workshop
From https://github.com/dt-kube-demo/catalog-service
 * [new branch]      master     -> jyarbrough-workshop/maste
new remote: jyarbrough-workshop
Done.
Cloning https://github.com/dt-kube-demo/customer-service
Forking customer-service to jyarbrough-workshop
Updating jyarbrough-workshop
From https://github.com/dt-kube-demo/customer-service
 * [new branch]      2          -> jyarbrough-workshop/2
 * [new branch]      master     -> jyarbrough-workshop/maste
new remote: jyarbrough-workshop
Done.
Cloning https://github.com/dt-kube-demo/front-end
```

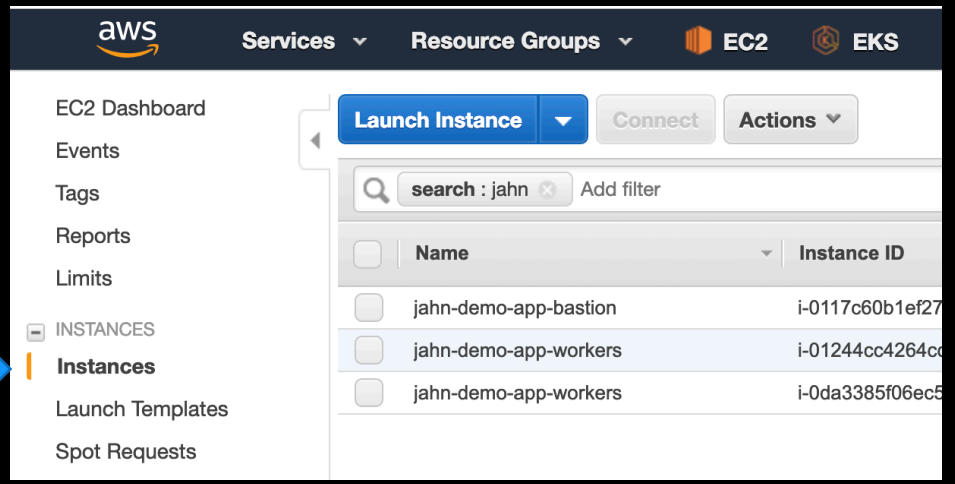


Verify cluster is up



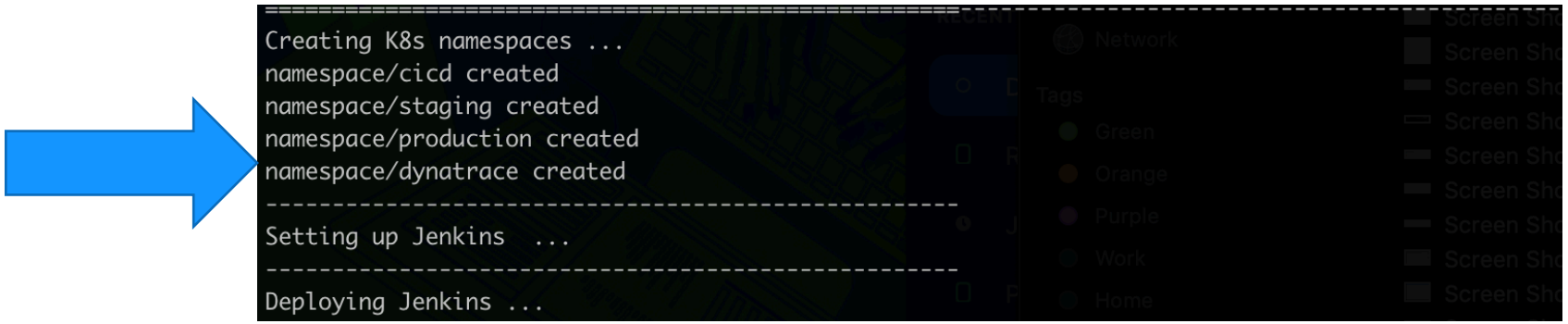


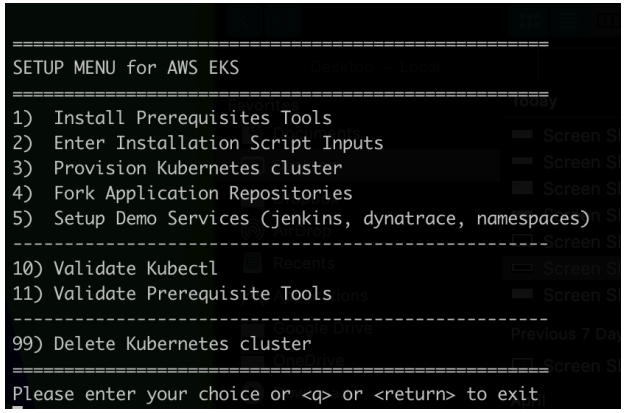


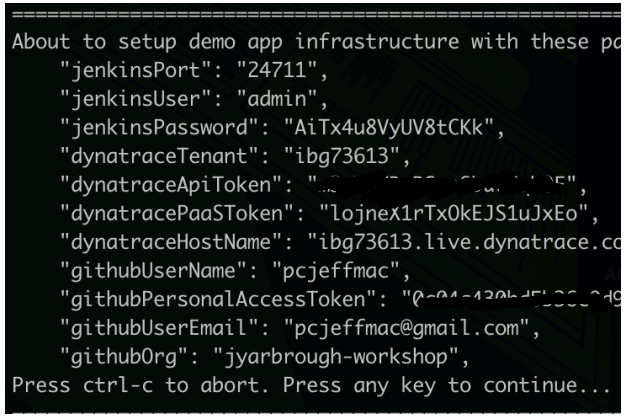


Run Setup Demo Services

1. Back on the bastion host, we now need to setup Jenkins, Dynatrace OneAgent Operator, and Kubernetes namespaces within the EKS cluster. To do this, run these commands:
   - cd ~/setup-infra/scripts
   - ./setup.sh eks
   - Select option 5) Setup Demo Services
   - Verify Inputs
2. This script will setup take several minutes.







Checkout Kubectl

Within terminal, you can run these commands to review Kubernetes

1. kubectl get ns
2. kubectl get nodes
3. kubectl get pods --all-namespaces

#1

```
NAME          STATUS   AGE
cicd          Active   3d
default       Active   6d
dynatrace     Active   3d
kube-public   Active   6d
kube-system   Active   6d
production    Active   3d
staging       Active   3d
```

#2

```
NAME                                           STATUS
ip-172-30-158-155.us-west-2.compute.internal   Ready
ip-172-30-169-183.us-west-2.compute.internal   Ready
```

#3

```
NAMESPACE     NAME                                          READY   STATUS
cicd          jenkins-deployment-5b7d94f5c8-bn7sl            1/1     Running
dynatrace     dynatrace-oneagent-operator-55db4d5856-fjw97   1/1     Running
dynatrace     oneagent-5fl9k                                1/1     Running
dynatrace     oneagent-xkq6q                                1/1     Running
kube-system   aws-node-2f6ps                                1/1     Running
kube-system   aws-node-bvhkv                                1/1     Running
kube-system   coredns-7d77776957-k2j8w                      1/1     Running
kube-system   coredns-7d77776957-nqnd7                      1/1     Running
kube-system   kube-proxy-t52sl                              1/1     Running
kube-system   kube-proxy-x8xfk                              1/1     Running
```

Configure Jenkins

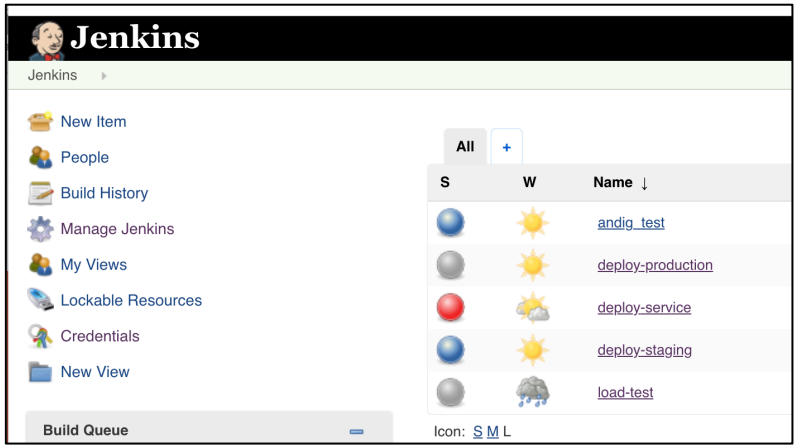
Log into Jenkins

1. To get the URL, run these commands in the terminal:
   - cd ~/setup-infra/scripts
   - ./helper.sh
   - Show Jenkins

```
--------------------------------------------------------------------------------
Jenkins is running @
http://ac2d1cdb8896611e9851d165c0af619d-520118970.us-east-1.elb.amazonaws.com:24711
Admin user          : admin
Admin password      : AiTx4u8VyUV8tCKk

NOTE: Credentials are from values in creds.json file
Password may not be accurate if you adjusted it in Jenkins UI
--------------------------------------------------------------------------------
```

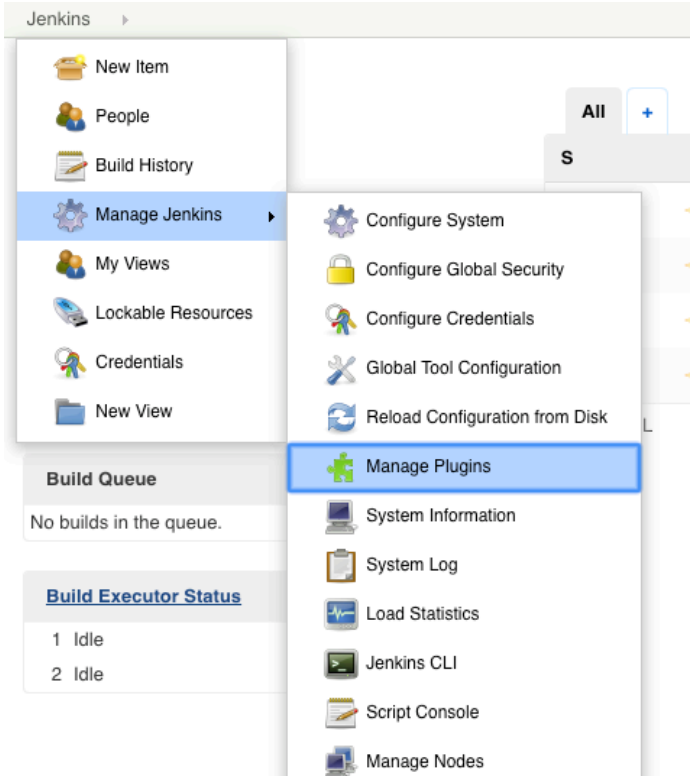
2. Then use the URL and credentials in browser to login

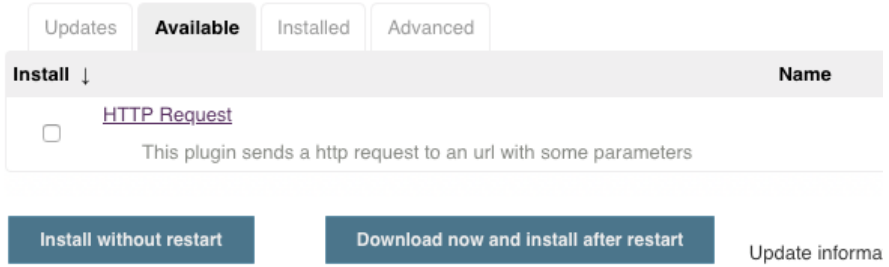
Login



Validate Jenkins

1. Install HTTP_Request plugin
   - Install plugin and let Jenkins restart



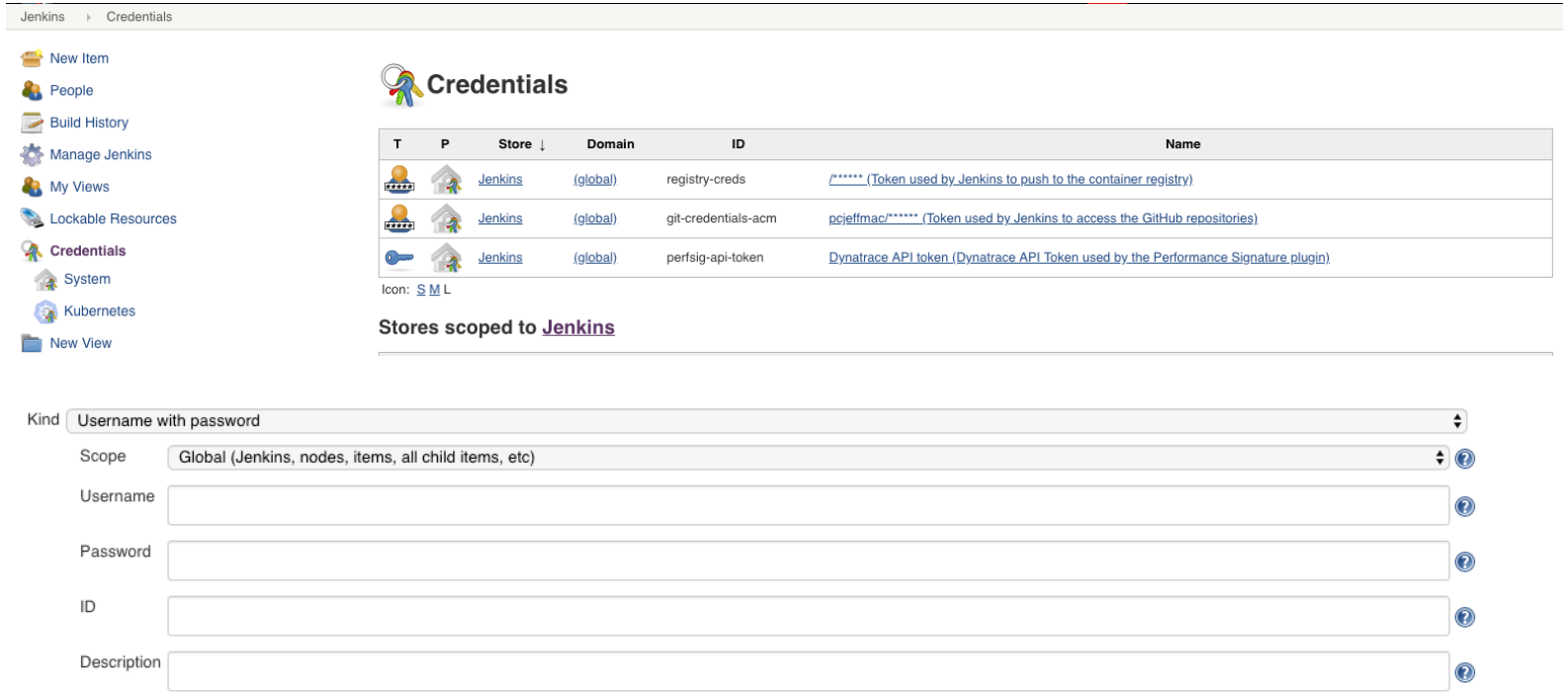


Validate Jenkins

1. Jenkins>Configure System
   - Test connection for Dynatrace API Token
   - Enter system admin email
   - Click Save

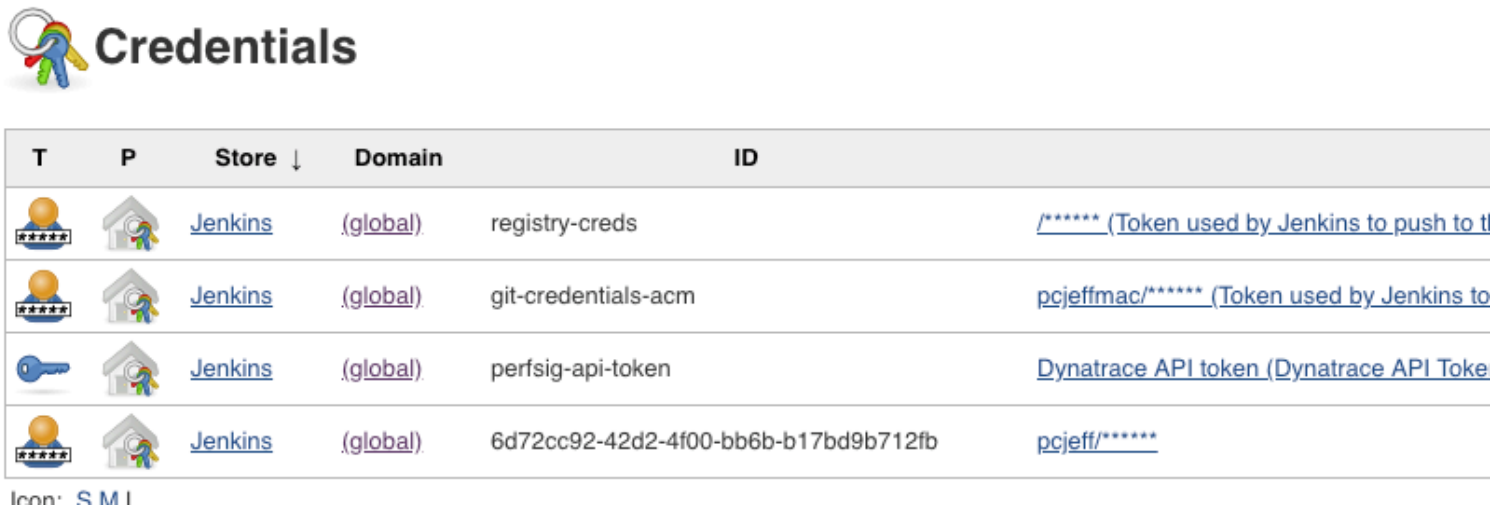
Validate Jenkins

1. Add a Credential
   - Credentials>add Credential
   - Kind username with password



Copy credential hash

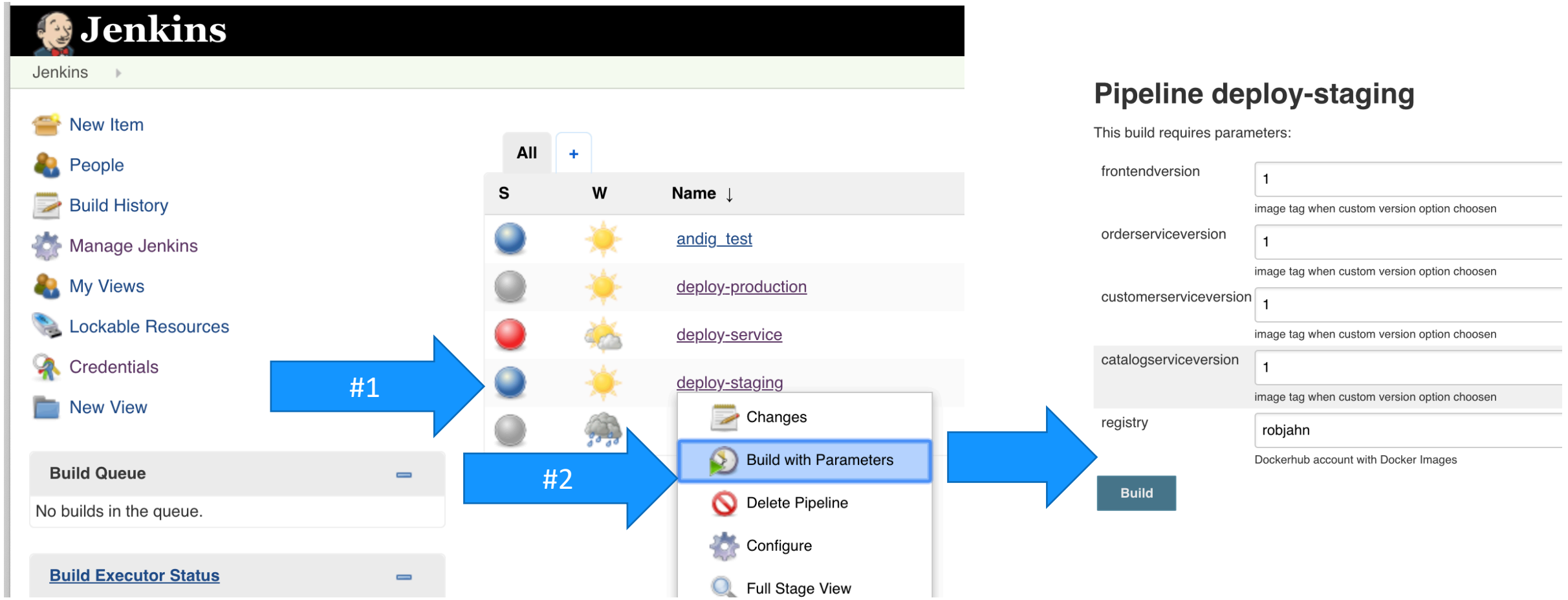
1. Select credentials and copy the ID.



Deploy the application

Lets run the “deploy-staging” job

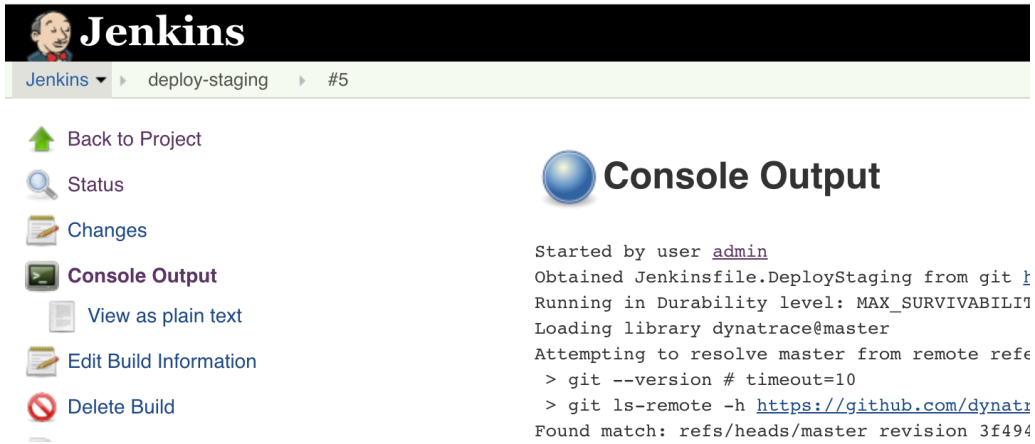
- Choose “build with parameters”, leave the versions all as “1” then click build



Lets run the “deploy-staging” job

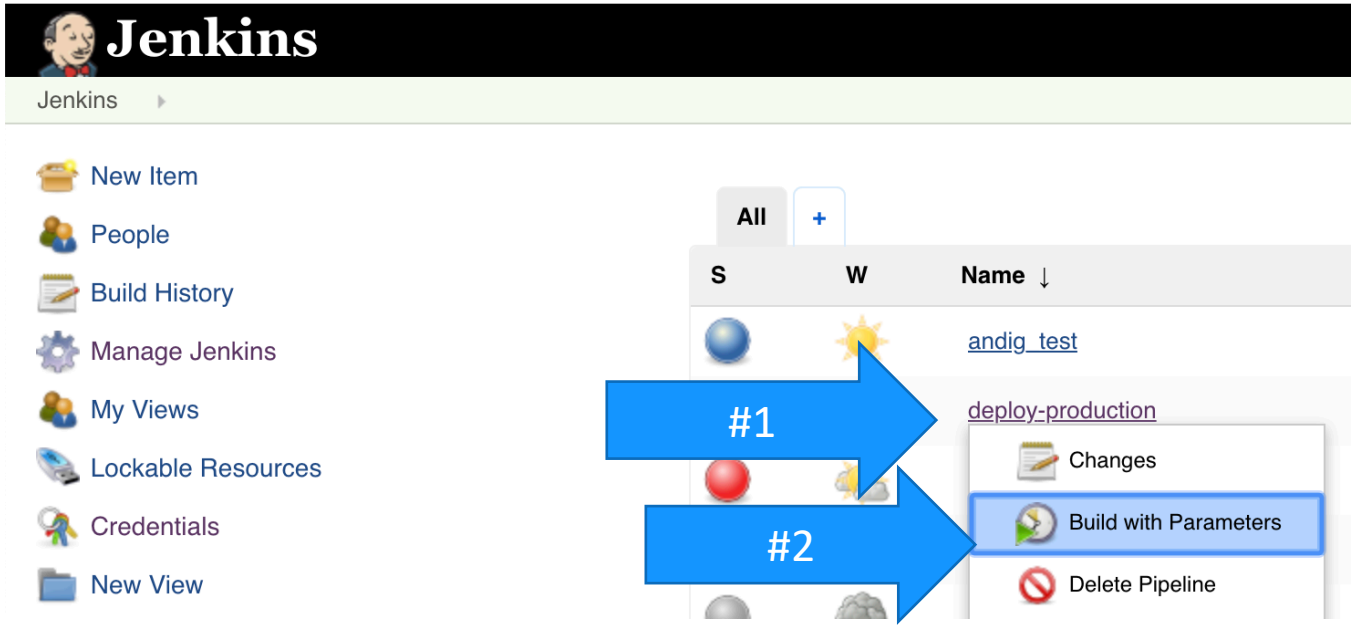
- View status and console log

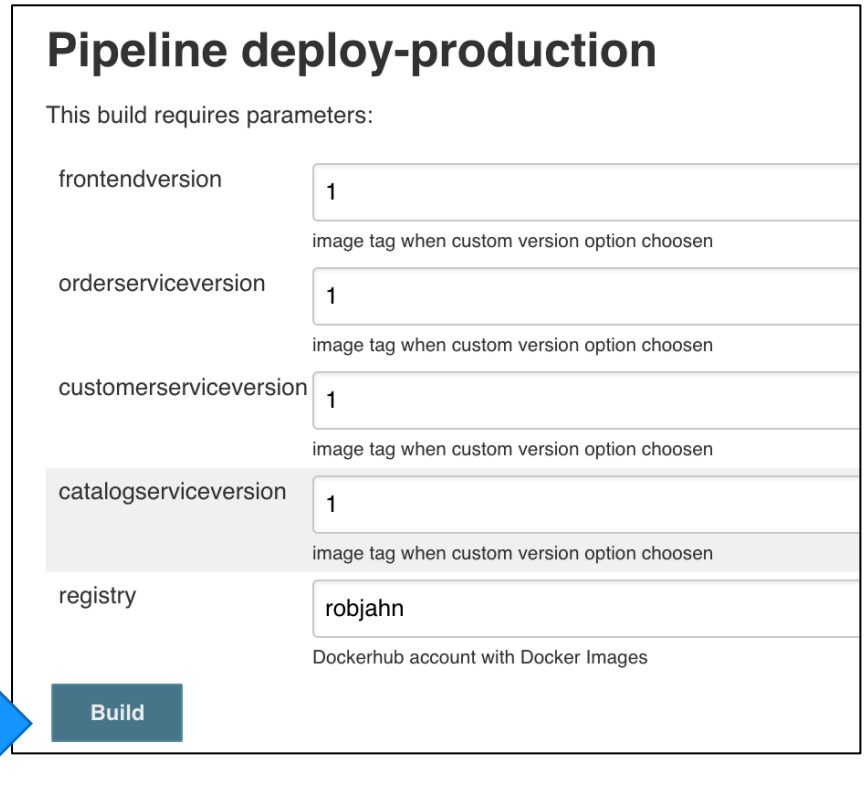
	Declarative: Checkout SCM	Deploy	Health Check	Push Dynatrace Deployment Event	Run e2e load test	quality gate
Average stage times: (Average full run time: ~5min 57s)	859ms	23s	1min 2s	20s	6min 0s	2s
#5 Mar 25 21:37 No Changes	225ms	24s	1min 2s	20s	4min 7s	1s



Lets run the “deploy-production” job

- Choose “build with parameters”



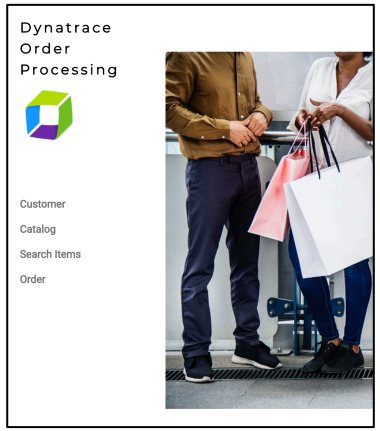


Log into bastion

1. To get the URL to the application, run these commands in the terminal:
   - cd ~/setup-infra/scripts
   - ./helper.sh
   - 1) show App

#1

2. Then use the URL to get to the demo app and navigate around



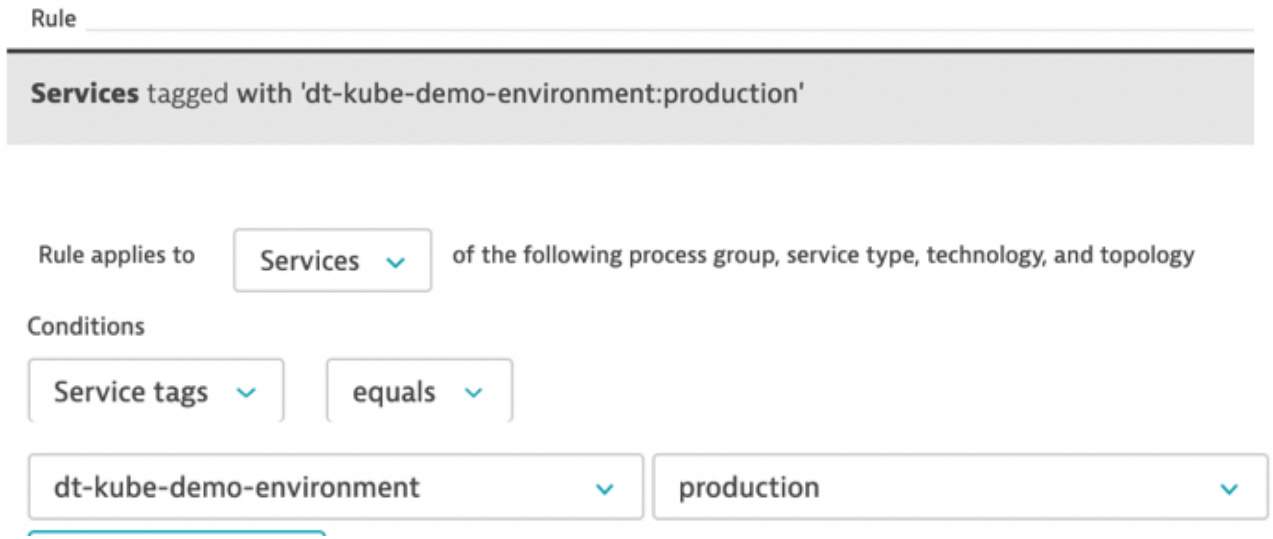
Explore & Finish Dynatrace Configuration

Dynatrace Review

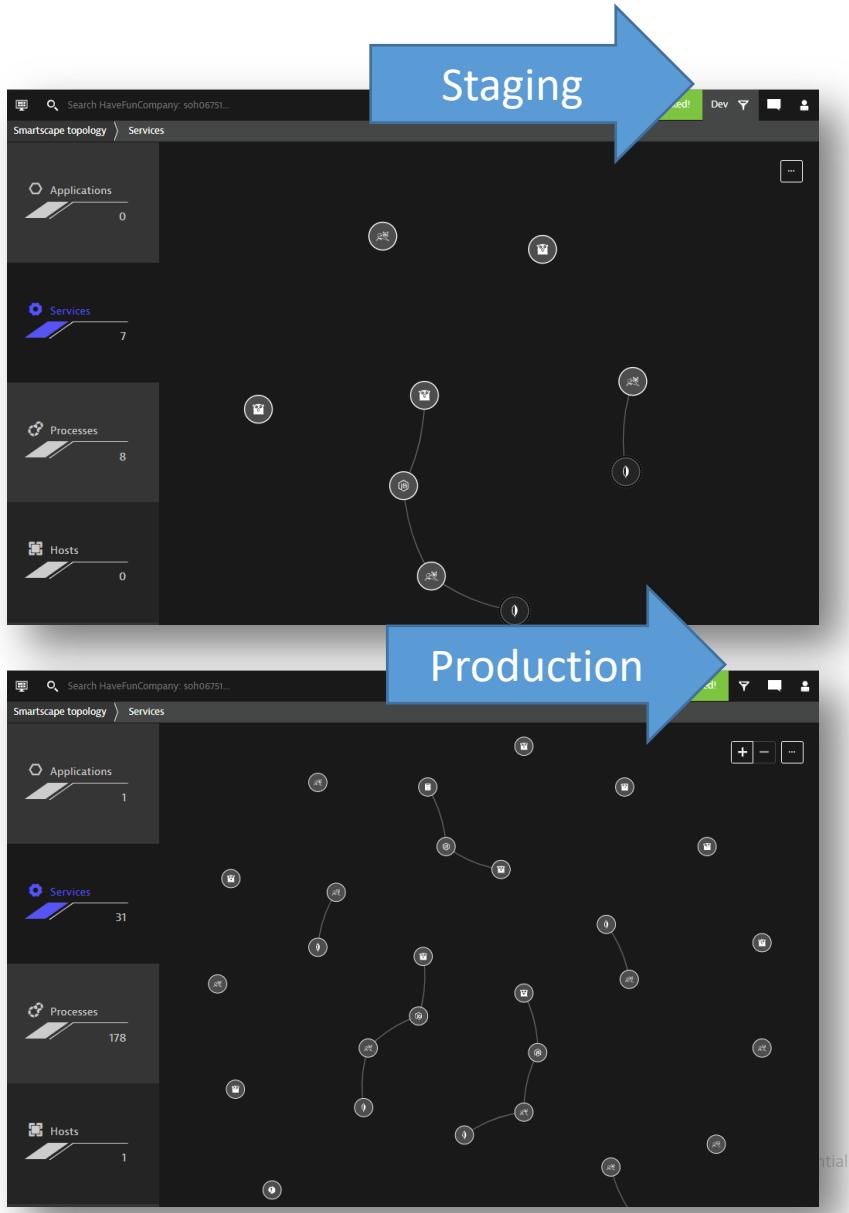
1. Hit the application UI a few times
2. Review smart scape and search for “front-end” - shows auto relationships
3. Review front-end service – tags
4. Review service flow
5. Review auto-tagging rules

Create Management Zones

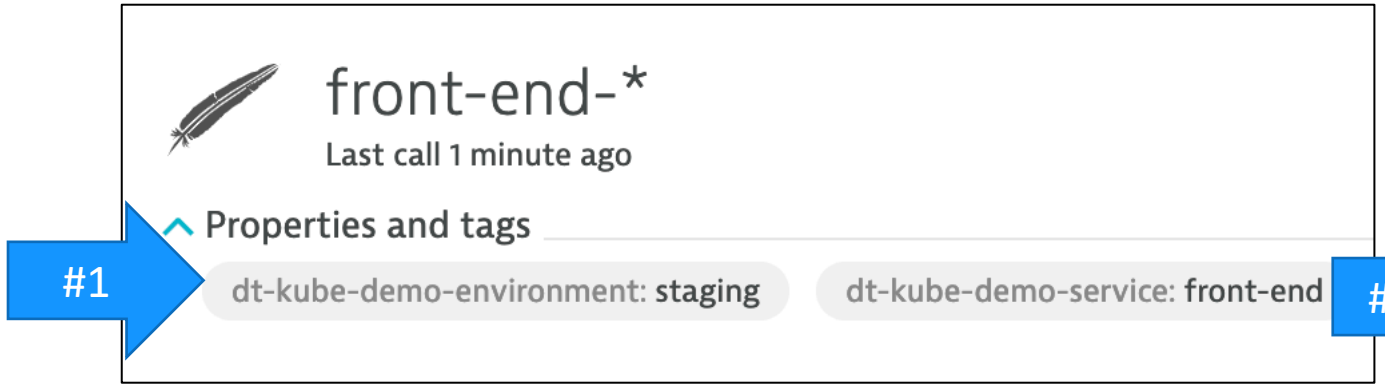
1. Under settings → Preferences → Management Zone choose ‘add new management zone’
2. Add new called ‘staging’ and add condition for ‘service-tag’ of ‘dt-kube-demo-environment = staging’. Click preview and save



3. Repeat for a new rule called ‘production’ with tag value = ‘production’



In Dynatrace Deployment Event shows up on Deployed Service



#1

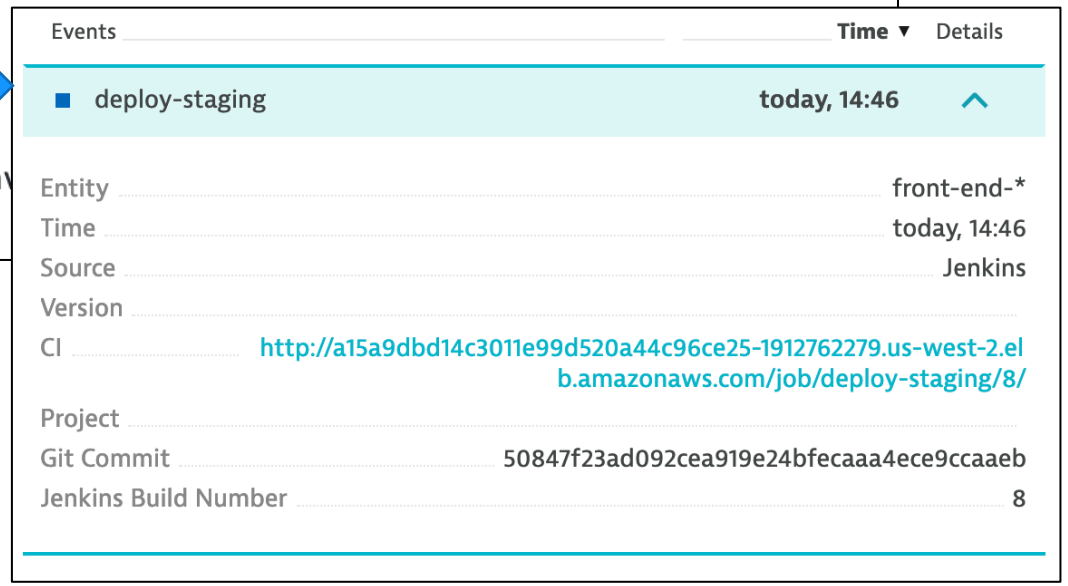
#2

```
def tagMatchRules = [
  [
    meTypes: [
      [meType: 'SERVICE']
    ],
    tags : [
      [ context: 'CONTEXTLESS', key: 'environment', value: 'staging' ]
    ]
  ]
```

#3

```
// push event using Jenkins plugin
// https://github.com/jenkinsci/performance-signature-dynatrace-plugin/blob/master/dynatrace/README.md
createDynatraceDeploymentEvent(
  envId: 'Dynatrace Tenant',
  tagMatchRules: tagMatchRules) {
    // do nothing. Need to do this after the wait as to hav
  }
```

#4



https://github.com/dt-kube-demo/deploy/blob/master/Jenkinsfile.DeployStaging

Add A Deployment event (http request)

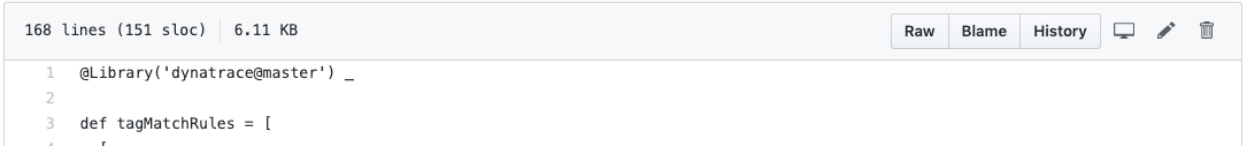
- Add Stage to git deploy.staging
  - We will need the credential ID
  - https://github.com/pcjeffmac/pipelineWorkshop/blob/master/buildrequest.txt
  - Copy credential ID into Authentication

#1

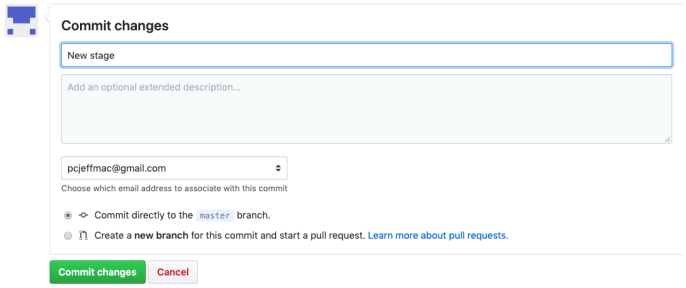
```
                }
            }"""
        //send json payload - create authentication credential in Jenkins
        httpRequest acceptType: 'APPLICATION_JSON',
        authentication: 'b06659fd-2cfe-48c6-80c6-a88f22cb1336',
        contentType: 'APPLICATION_JSON',
        customHeaders: [[maskValue: true, name: 'Authorization',
        value: "Api-Token ${DT_API_TOKEN}"]],
        httpMode: 'POST',
        ignoreSslErrors: true,
        requestBody: body,
        responseHandle: 'NONE',
        url: "${DT_TENANT_URL}/api/v1/events/"
        }
```

Add A Deployment event (http request)

- Add Stage to **Jenkinsfile.DeployStaging**
- Copy buildrequest.txt and past in github repo.
  - Go to Org repo>file
  - Click edit this file
  - Paste stage after
    - “Push Dynatrace Deployment Event” stage,
  - Commit Change

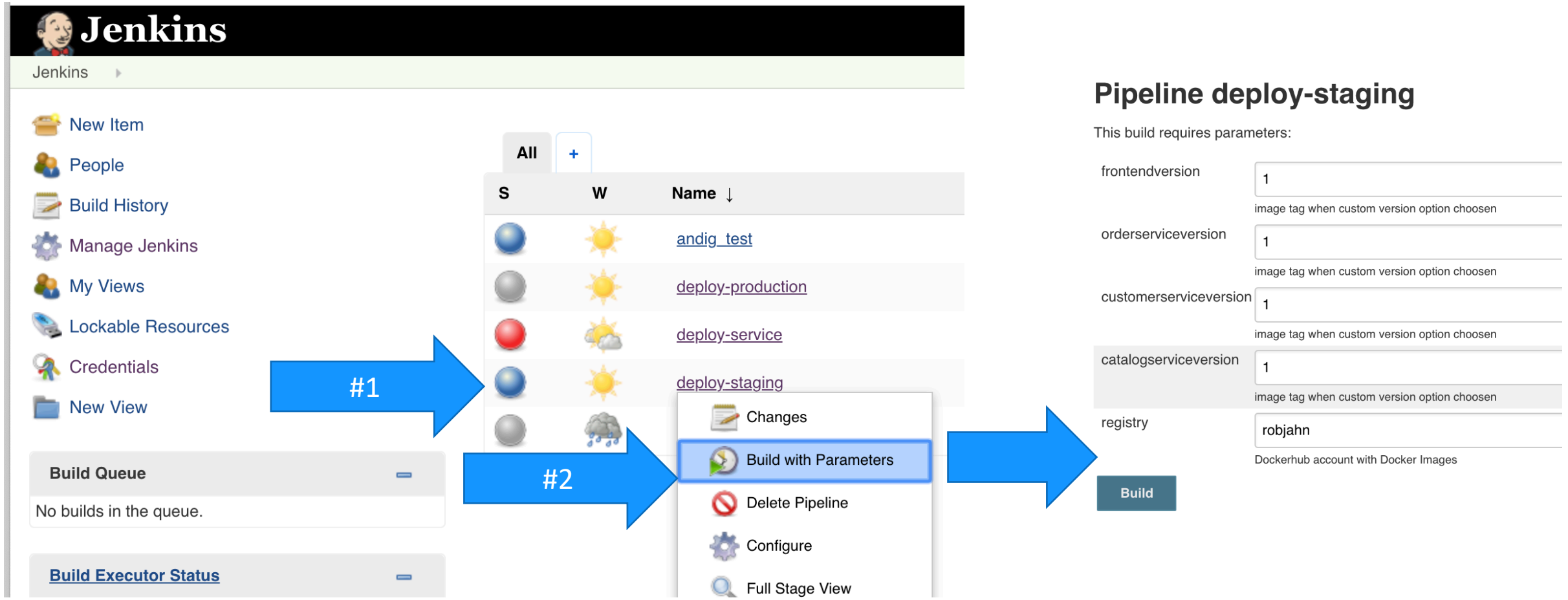



```
    stage('Push Dynatrace Deployment Event') {
      steps {
        script {
          echo "Waiting for the service to be tagged..."
          sleep 20

          // push event using Jenkins plugin
          // https://github.com/jenkinsci/performance-signature-dynatrace-plugin/blob/master/dynatrace/README.md
          createDynatraceDeploymentEvent(
            envId: 'Dynatrace Tenant',
            tagMatchRules: tagMatchRules,
            customProperties: [[key: 'frontendversion', value: "${params.frontendversion}"],[key: 'orderversion', value: "${par
          ) {
              // do nothing. Need to do this after the wait as to have service up
            }
        }
      }
    }

stage('Event-Post-Service-Deployment') { //Dynatrace POST action for deployment Event - requires HTTP Request Plugin
  steps {
    script {
      def body = """{"eventType": "CUSTOM_DEPLOYMENT",
      "attachRules": {
        "tagRule" : {
        "meTypes" : "SERVICE",
        "tags" : [
          {
          "context" : "CONTEXTLESS",
          "key" : "dt-kube-demo-service",
          "value" : "front-end"
          }
          ]
        }
```

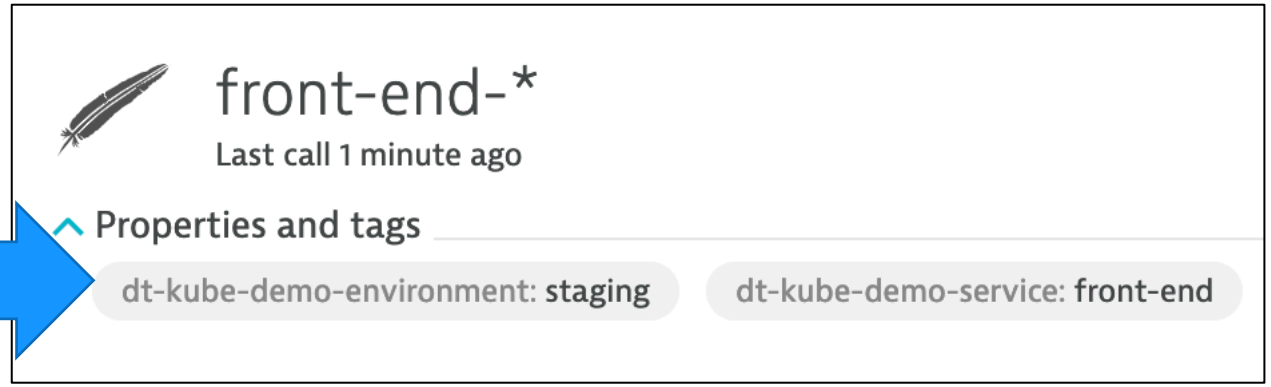


Lets run the “deploy-staging” job

- Choose “build with parameters”, leave the versions all as “1” then click build



In Dynatrace Deployment Event shows up on Deployed Service

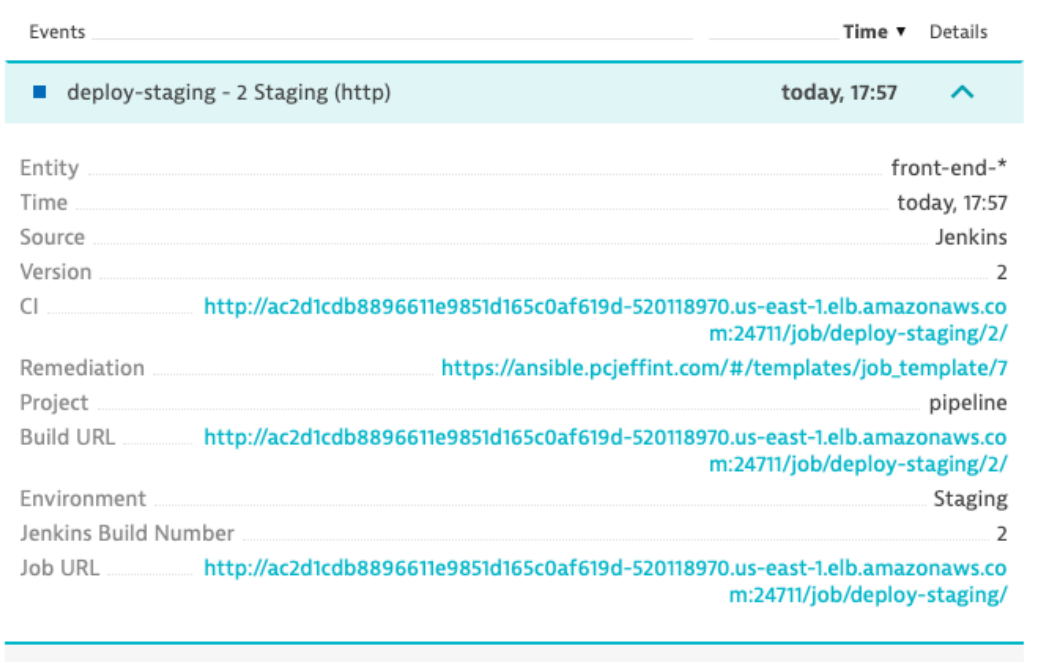


#1

#2

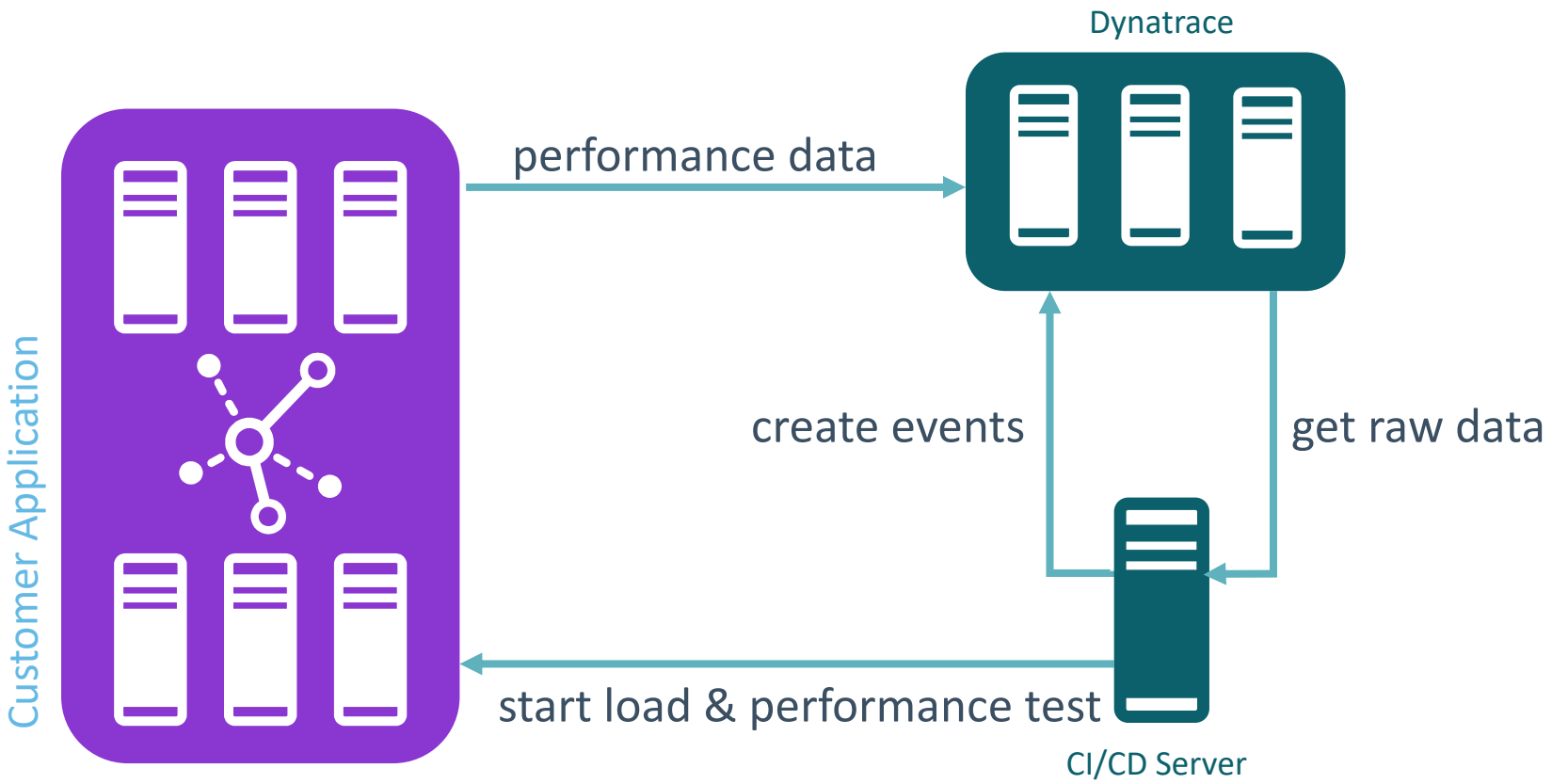
```
stage('Event-Post-Service-Deployment') { //Dynatrace POST action for deployment Event - requires HTTP Request Plugin
        steps {
                script {
                        def body = """{"eventType": "CUSTOM_DEPLOYMENT",
                        "attachRules": {
                                "tagRule" : {
                                "meTypes" : "SERVICE",
                                "tags" : [
                                        {
                                        "context" : "CONTEXTLESS",
                                        "key" : "dt-kube-demo-service",
                                        "value" : "front-end"
                                        }
                                        ]
                                }
                        },
```

#3

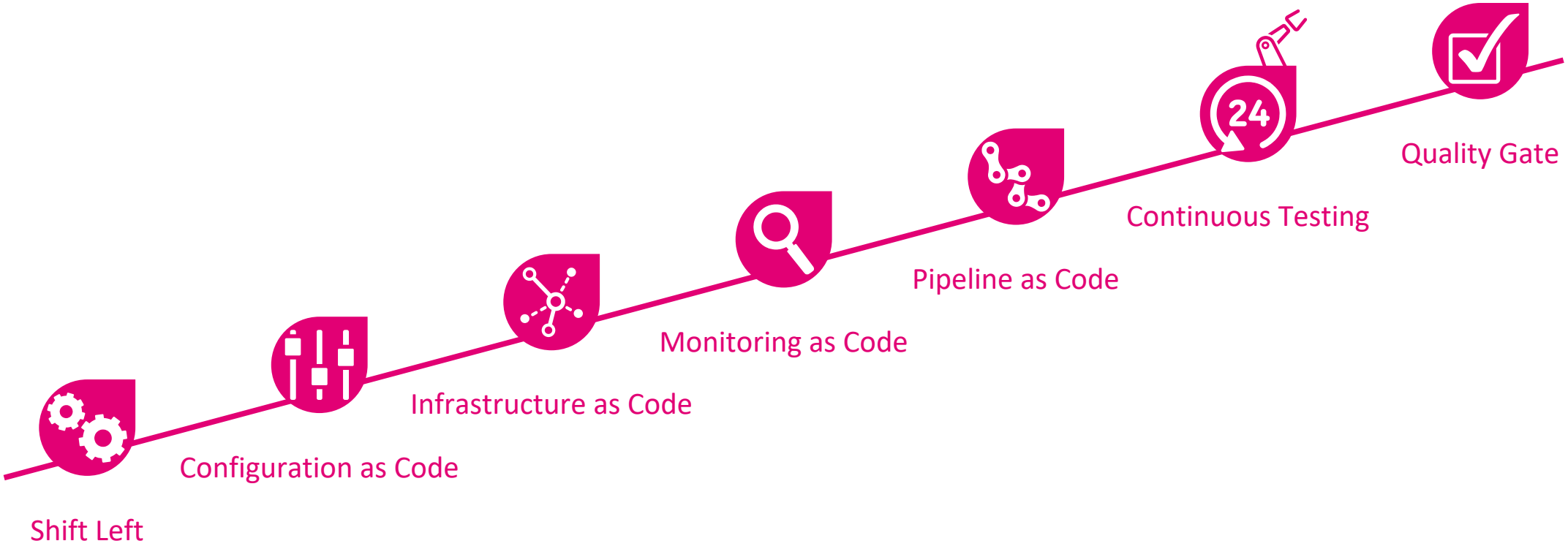


https://github.com/dt-kube-demo/deploy/blob/master/Jenkinsfile.DeployStaging

Performance Signature: Architecture



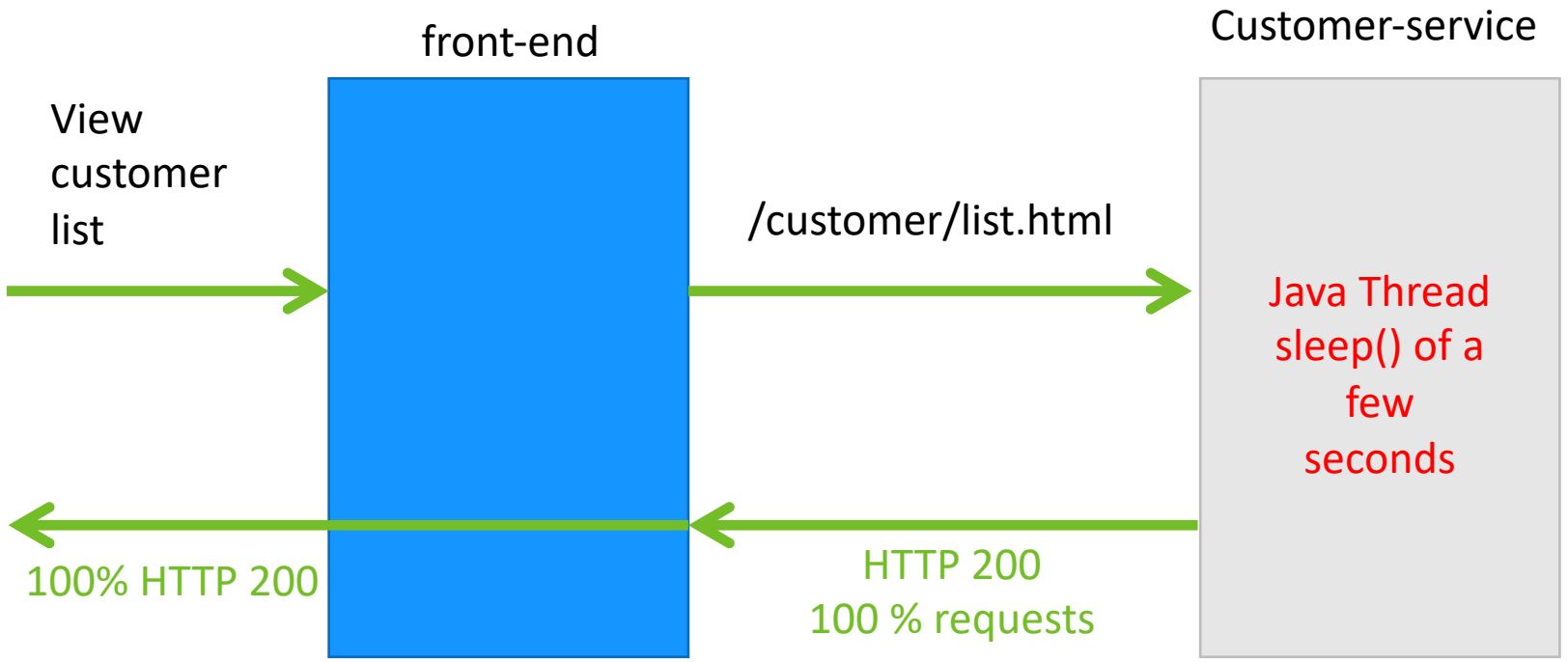
Performance Signature: Concepts



Lets prevent an issue

Quality Gate

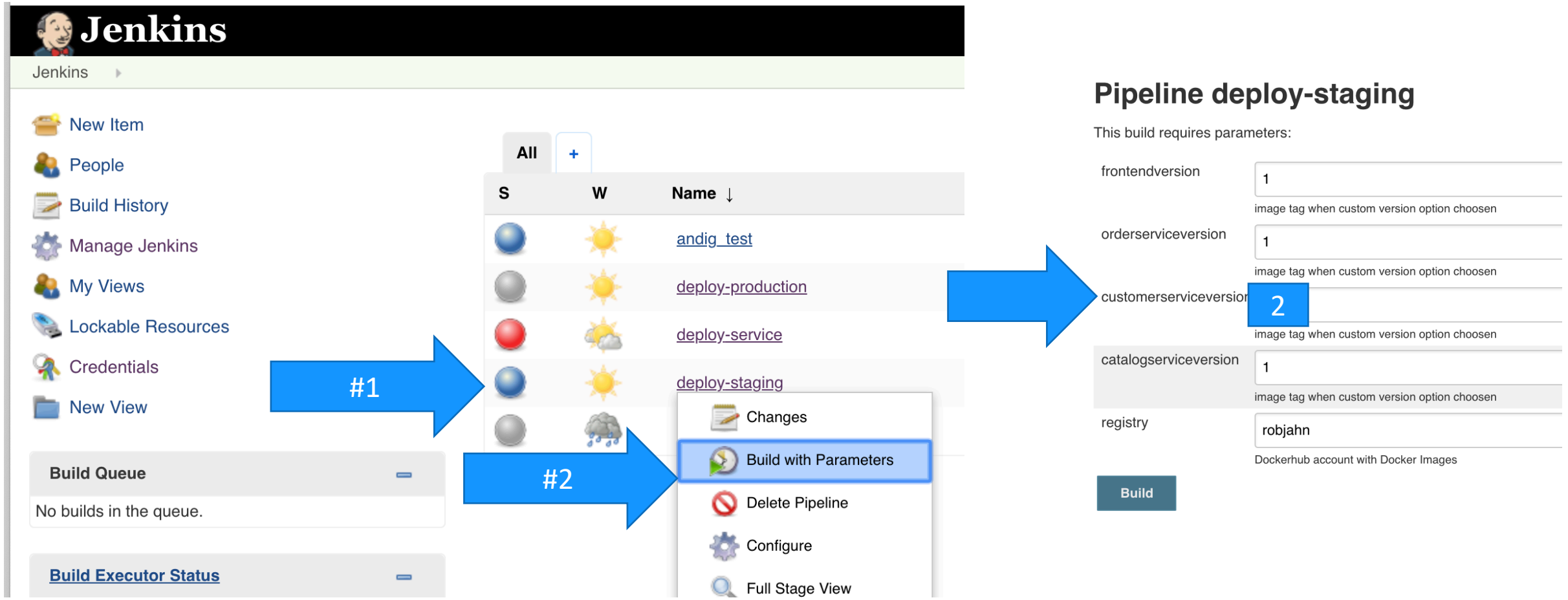
Scenario 1: Response time issues



- Through just test metrics we would not detect the problem to be the backend
- Metrics would tell us Frontend is slow
- ONLY Tracing data can tell us that the problem only happens on the service call

Lets run the “deploy-staging” job

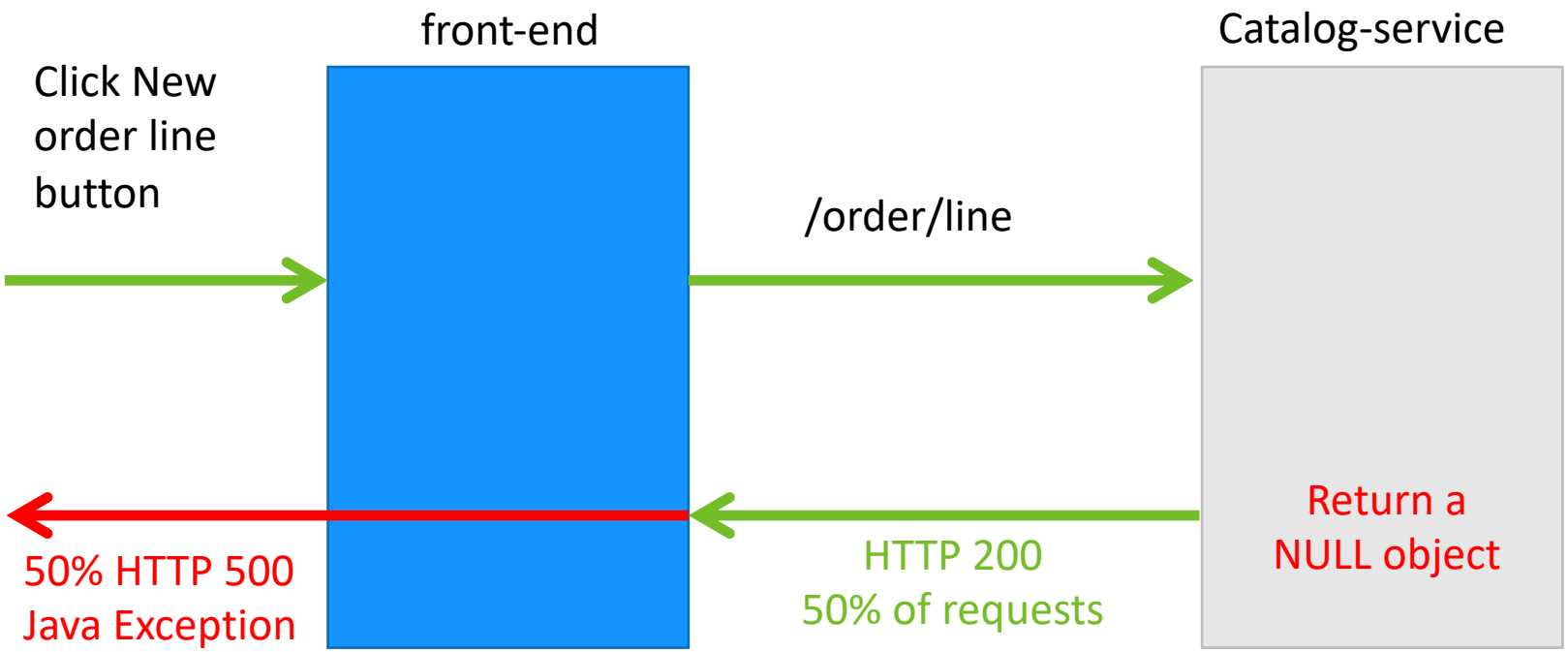
- Choose “build with parameters” and select version = ”2” for the customer service then click build



Review

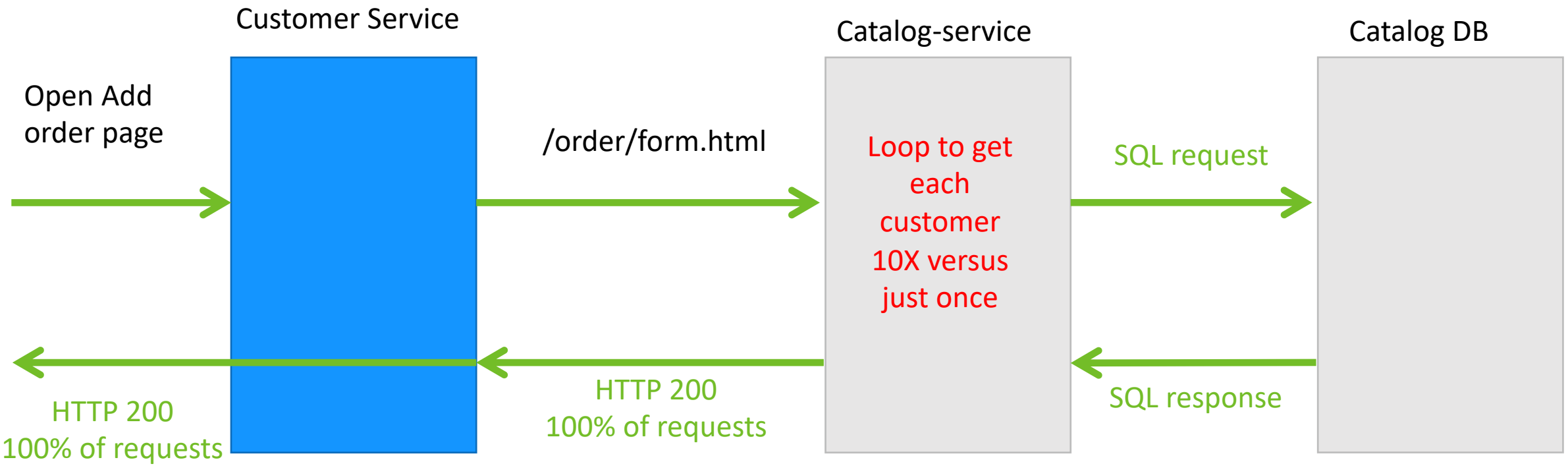
1. Review DT as test runs
2. Review the monspec file
3. Review the perf signature output in Jenkins

Scenario 2: Backend has exceptions in the code



- Through just test metrics we would not detect the problem to be the backend
- Metrics would tell us Frontend has exceptions
- ONLY Tracing data can tell us that the problem only happens on the service call

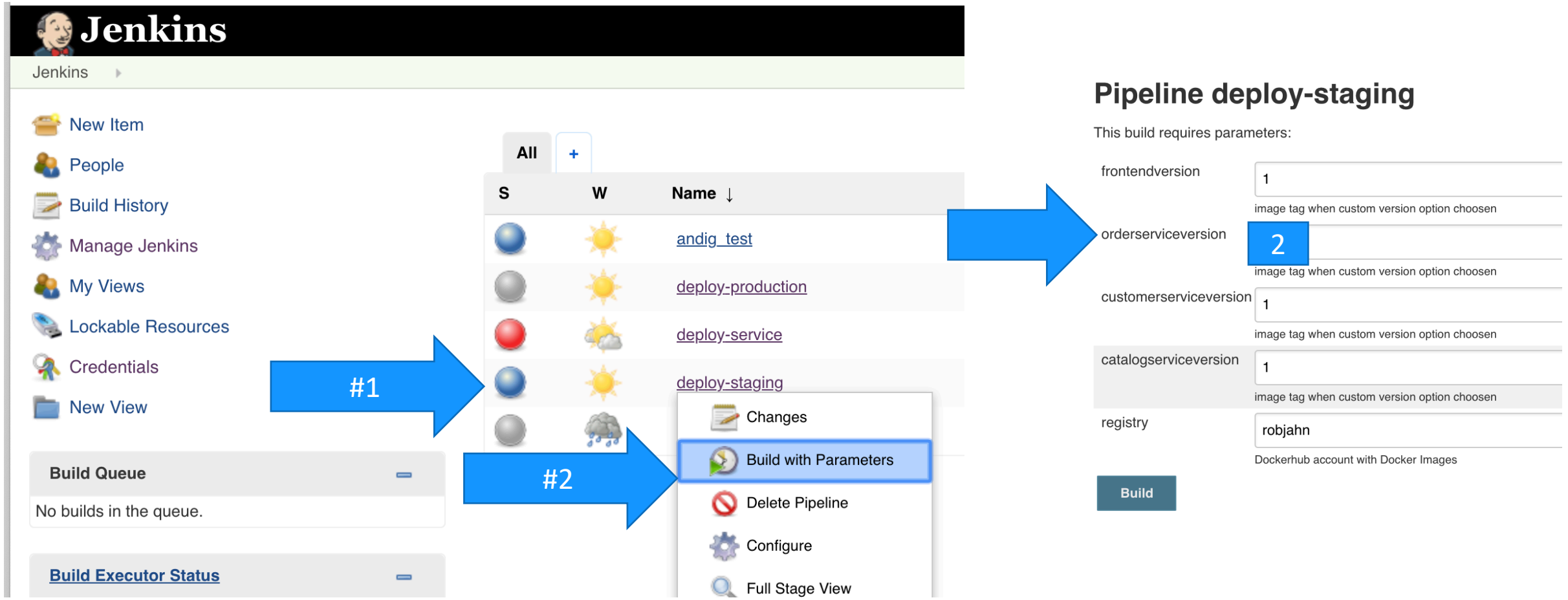
Scenario 3: N+1 Pattern



- Metrics show everything is OK with no responce time issues
- Only Tracing can give you what the SQL processing was doing

Lets run the “deploy-staging” job

- Choose “build with parameters” and select version = ”2” for the order service then click build



Review

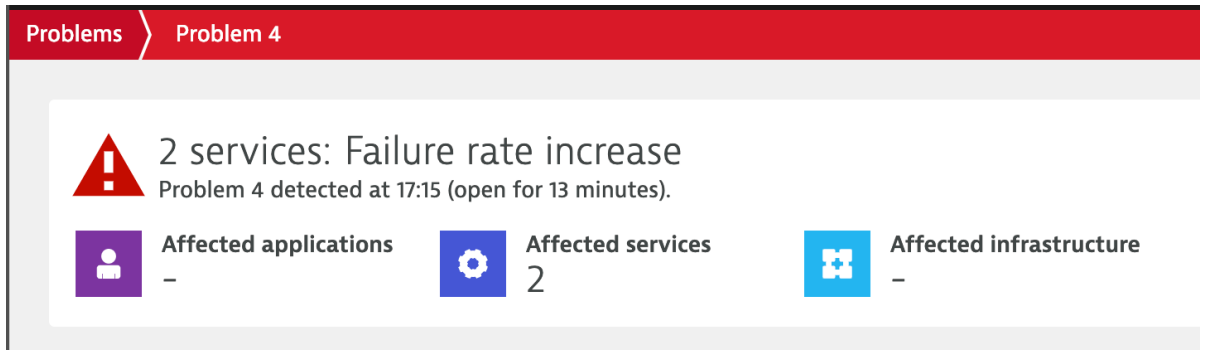
1. Review DT as test runs
2. Review the perf signature output in Jenkins
3. Review the monspec file

Automated Operations

Self-Healing

Self-healing Automation

1. Configure a Dynatrace problem notification custom webhook for production
2. Remediation is to run the Jenkins pipeline to re-deploy a “good” version of the app
3. Generate some continuous load against production (simulate users traffic)
4. Manually Release a “bad” version of code --- Dynatrace will detect and trigger the web hook



1 Dynatrace problem notification webhook

Problem indicates impacted entities

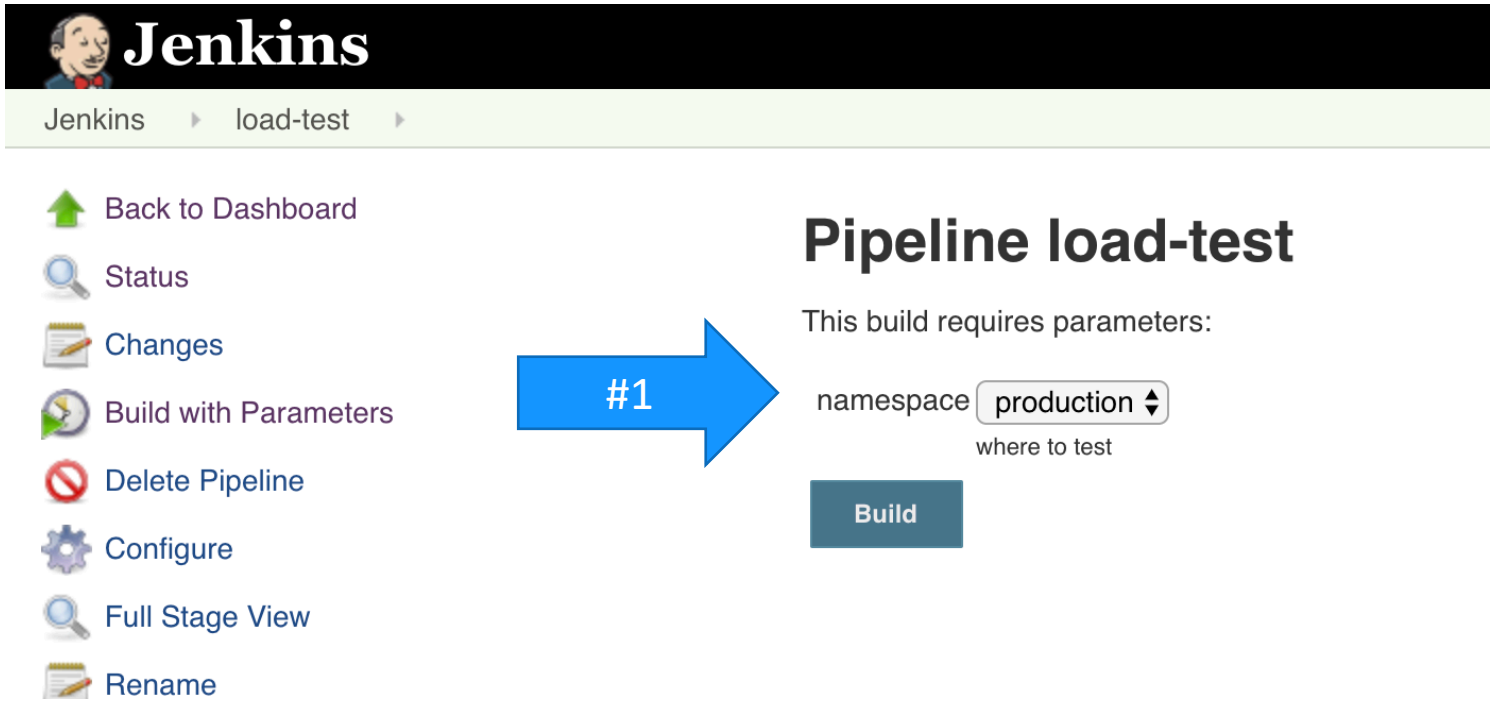
Jenkins

2 Deploy “good” version

Remediate issue

Run the continuous “load-test” job in production

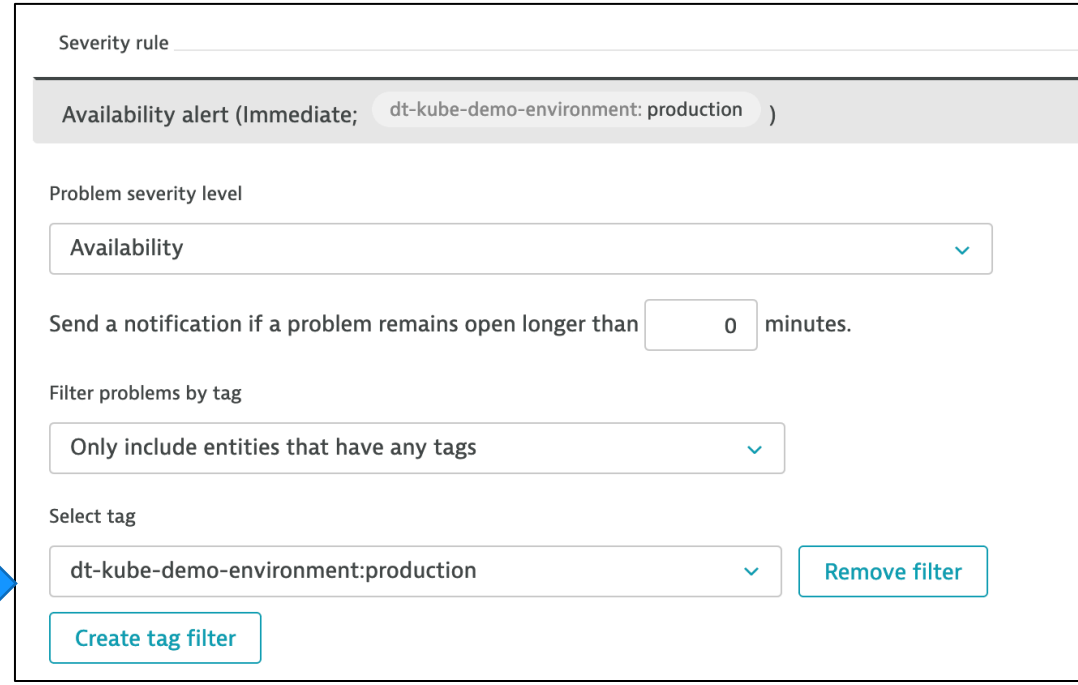
- Choose “build with parameters” and select “production” for the namespace then click build
- This job will just run until you cancel the job



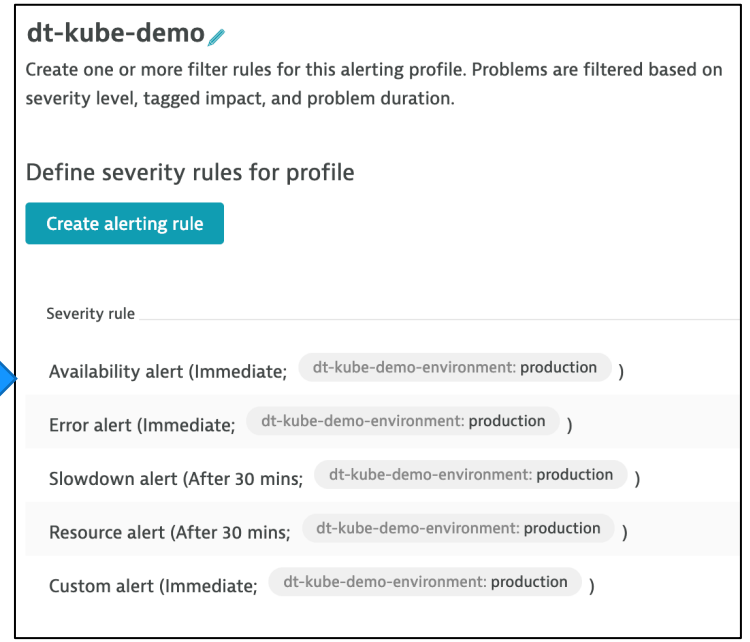
Add an alerting profile for production

- Goto settings → alerting → add a new profile
  - Name = dt-kube-demo
- Within each rule, add a filter:
  - dt-kube-demo-environment:production
- Repeat for each rule and save

#1



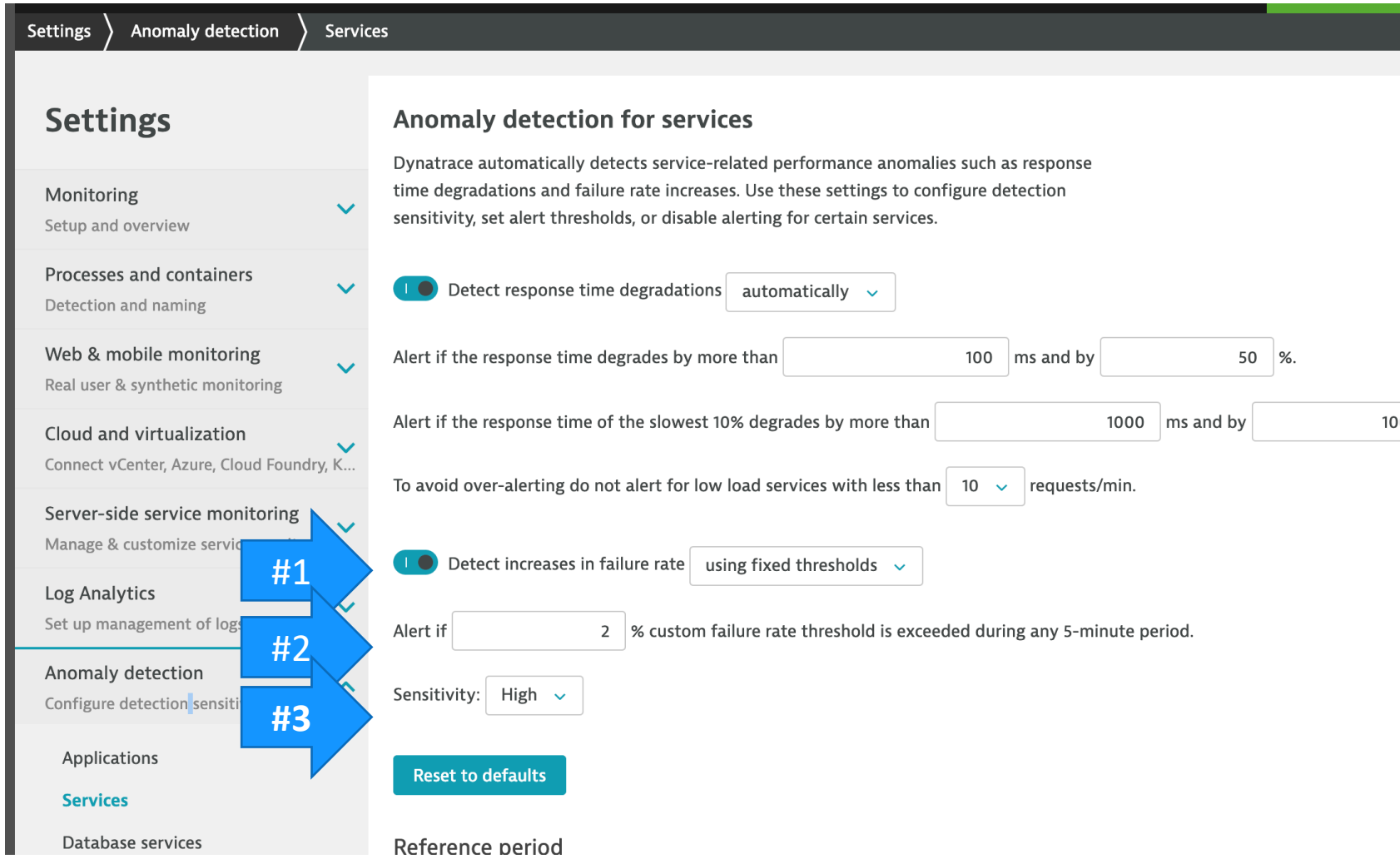
#2



Adjust anomaly detection in Dynatrace

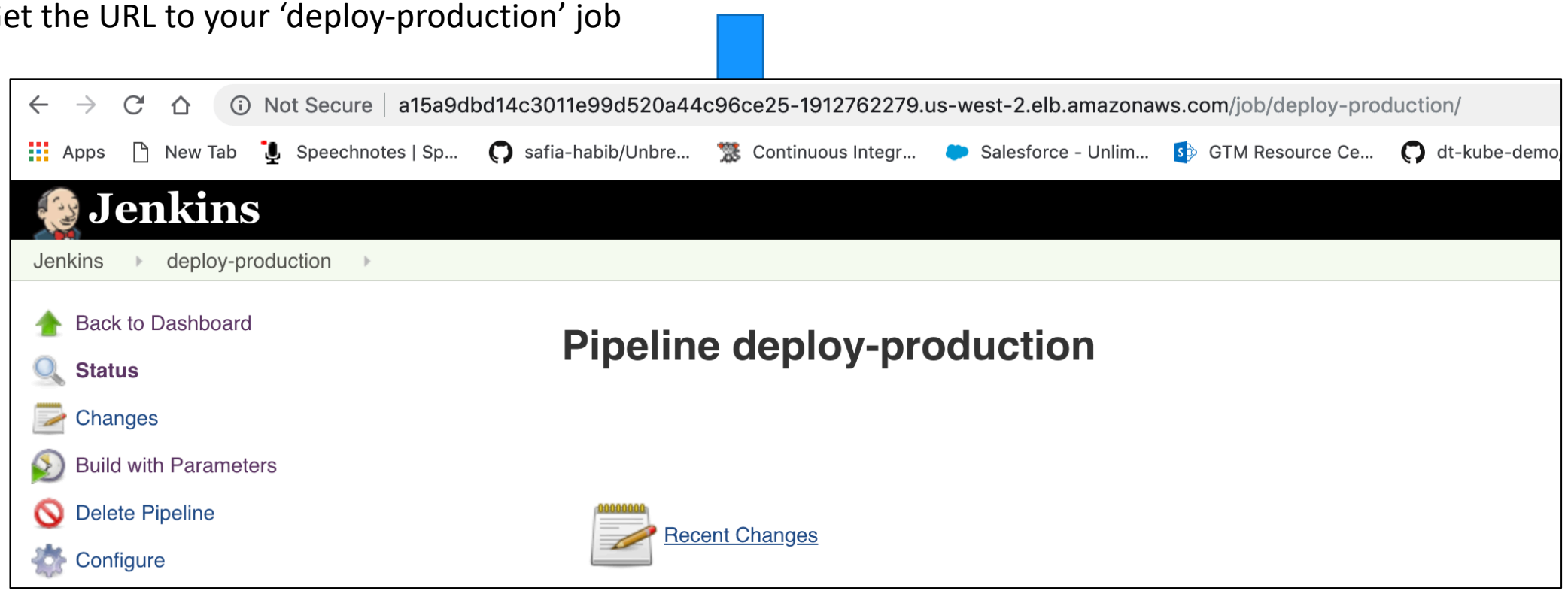
Goto settings → anomaly detection → services

1. Adjust failure to “using fixed thresholds”
2. Set “alert if” to 2%
3. Set “sensitivity” to high



Dynatrace problem notifications: Step 1 of 3

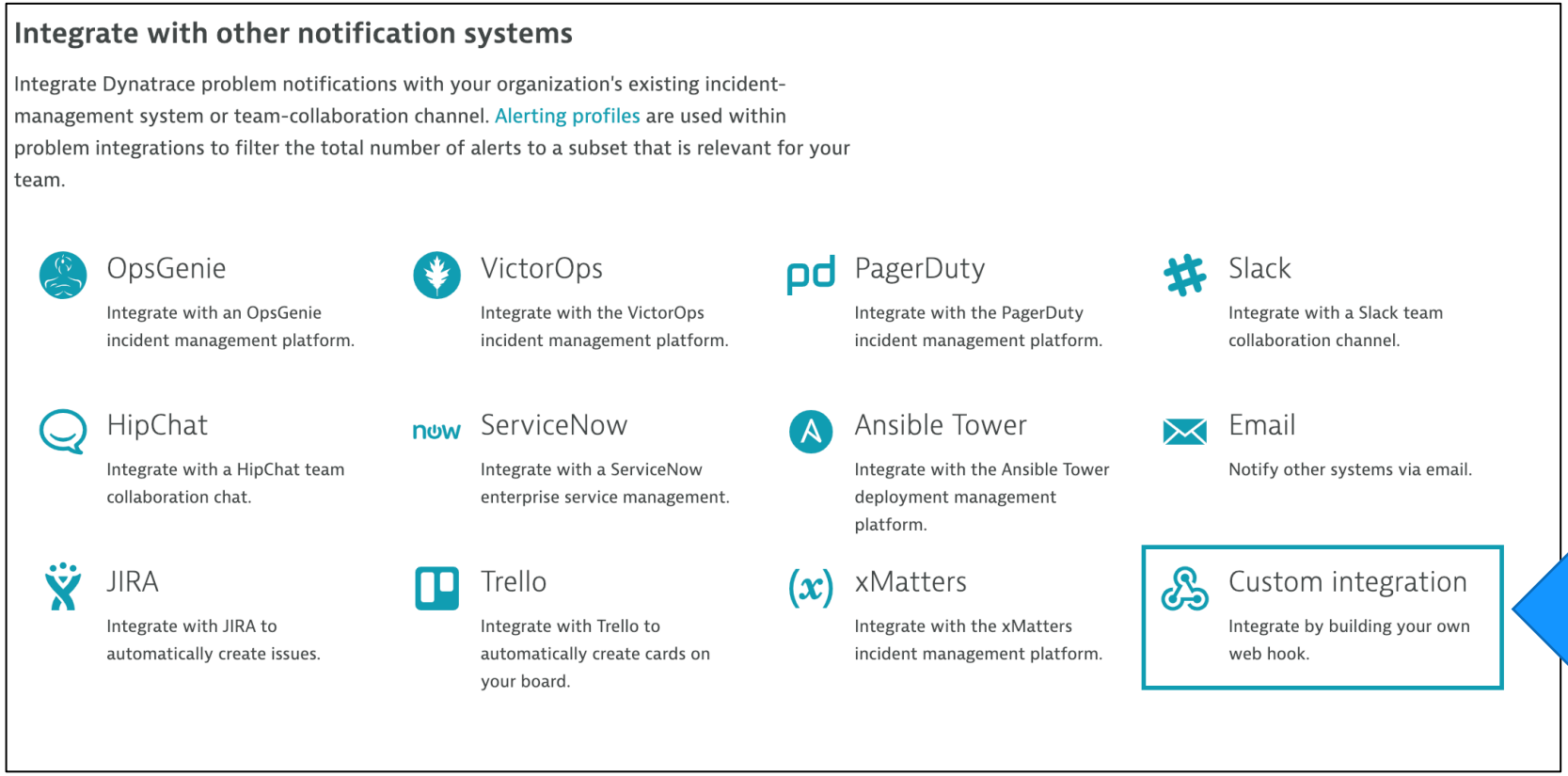
Get the URL to your ‘deploy-production’ job



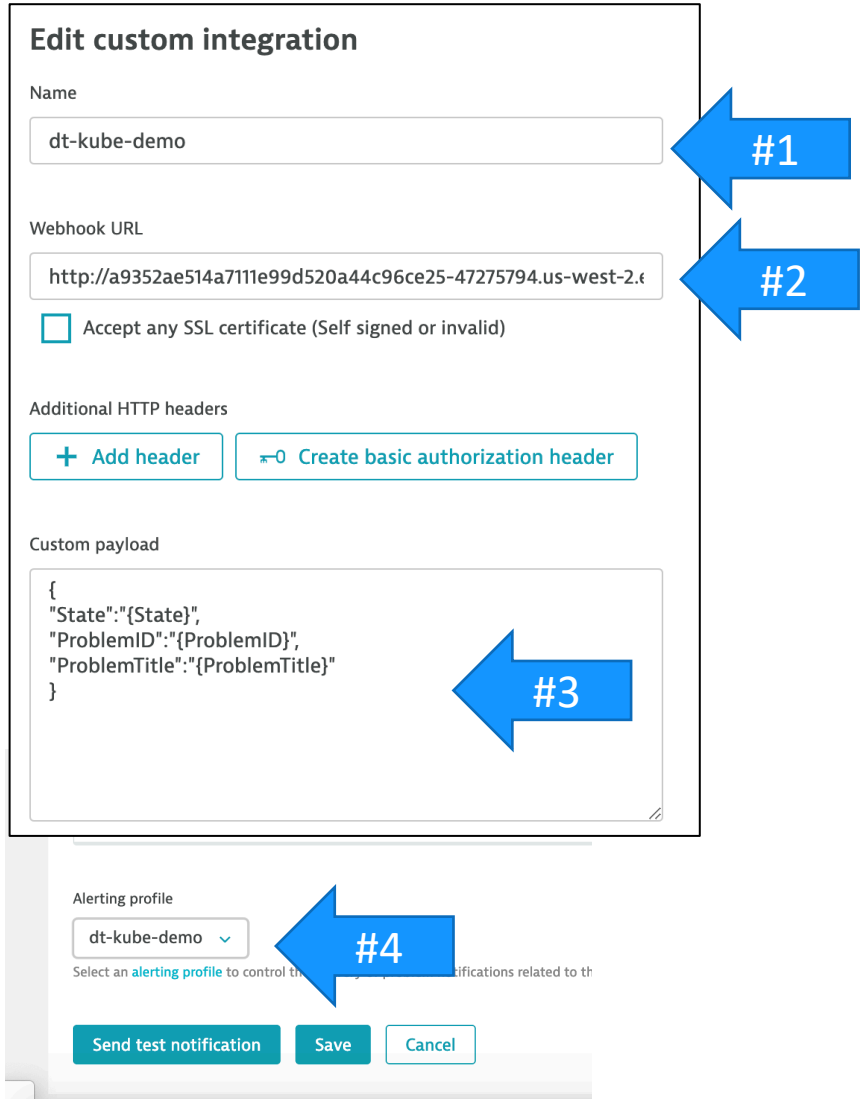
For example: http://a9352ae514a7111e99d520a44c96ce25-47275794.us-west-2.elb.amazonaws.com/job/deploy-production

Dynatrace problem notifications: Step 2 of 3

1. In Dynatrace, navigate to settings → Integration → Problem notifications
2. Click on Custom Integration

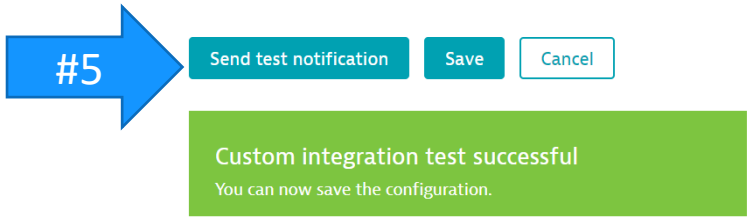


Dynatrace problem notifications: Step 3 of 3



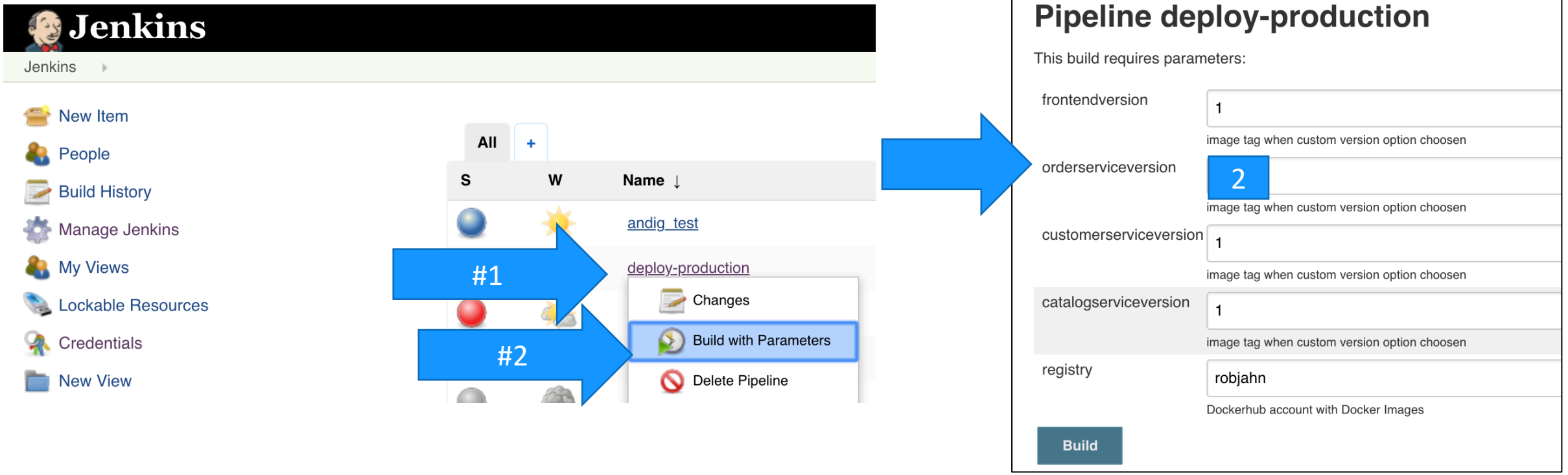
1. Go to Dynatrace and navigate to Settings->Integration->Problem notifications->Setup Notification and Choose ‘Custom Integration’. Name = “dt-kube-demo”
2. Add the Selfhealing Function App URL. The part in yellow if from your Jenkins ‘deploy-production’ job URL. Add suffix in blue:

   http://a9352ae514a7111e99d520a44c96ce25-47275794.us-west-2.elb.amazonaws.com/job/deploy-production/buildWithParameters?token=DT&cause=auto-remediation-from-Dynatrace
3. Leave Custom payload as the default.
4. **At the bottom, pick the ”dt-kube-demo” alert profile we just added**
5. Test your connection and confirm success. NOTE: This will start a ‘deploy-production’ job



Lets run the “deploy-production” job to simulate a problem build

- Choose “build with parameters” and select version = ”2” for the order service



Review

1. Watch Dynatrace for the problem to appear
2. Monitor Jenkins for the remediation job to start
3. Watch Dynatrace when remediation is applied
4. Watch Dynatrace close the problem after three samples periods (~15 minutes) once Dynatrace automatically validates the problem resolution

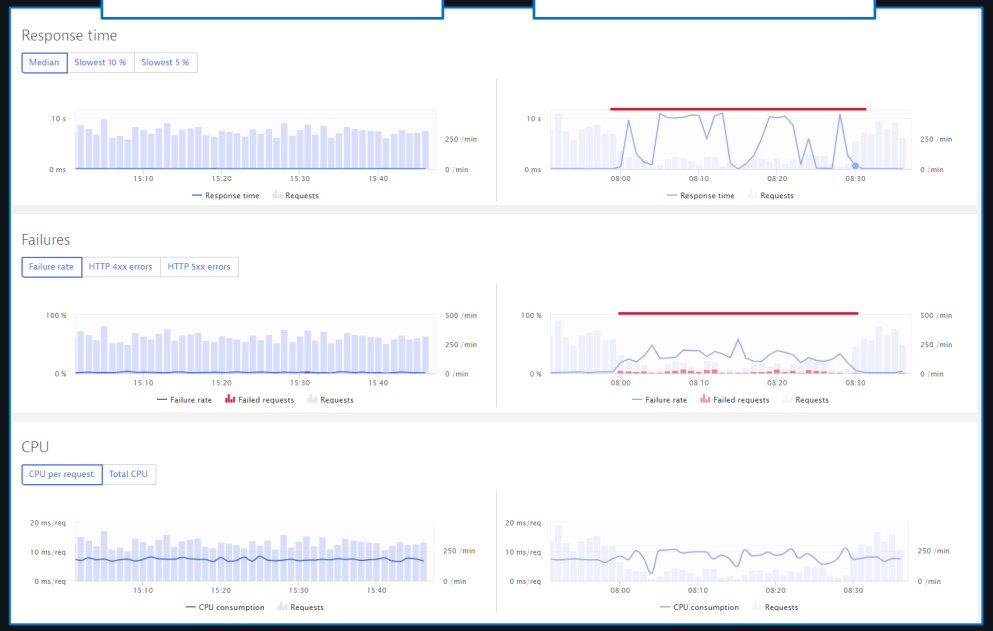
Recap

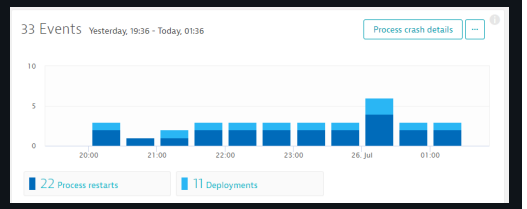
Unbreakable delivery pipeline in action

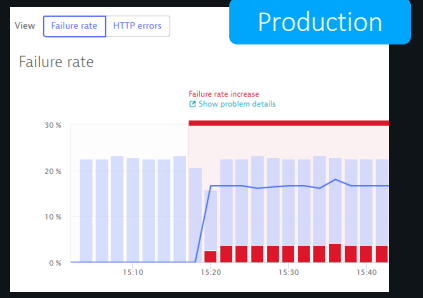
1 Staging → 2 Approve Staging → 3 Production → 4 Approve Production → 5

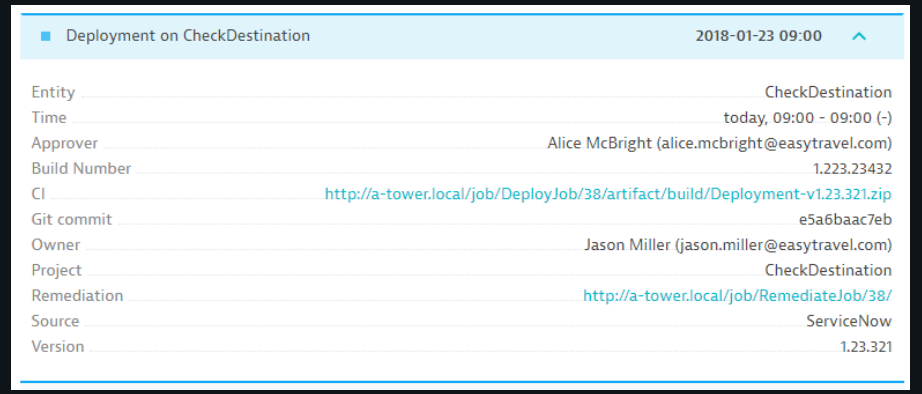
Build #17

Build #18

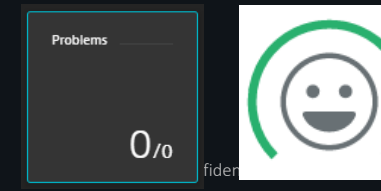








Auto-Remediate!

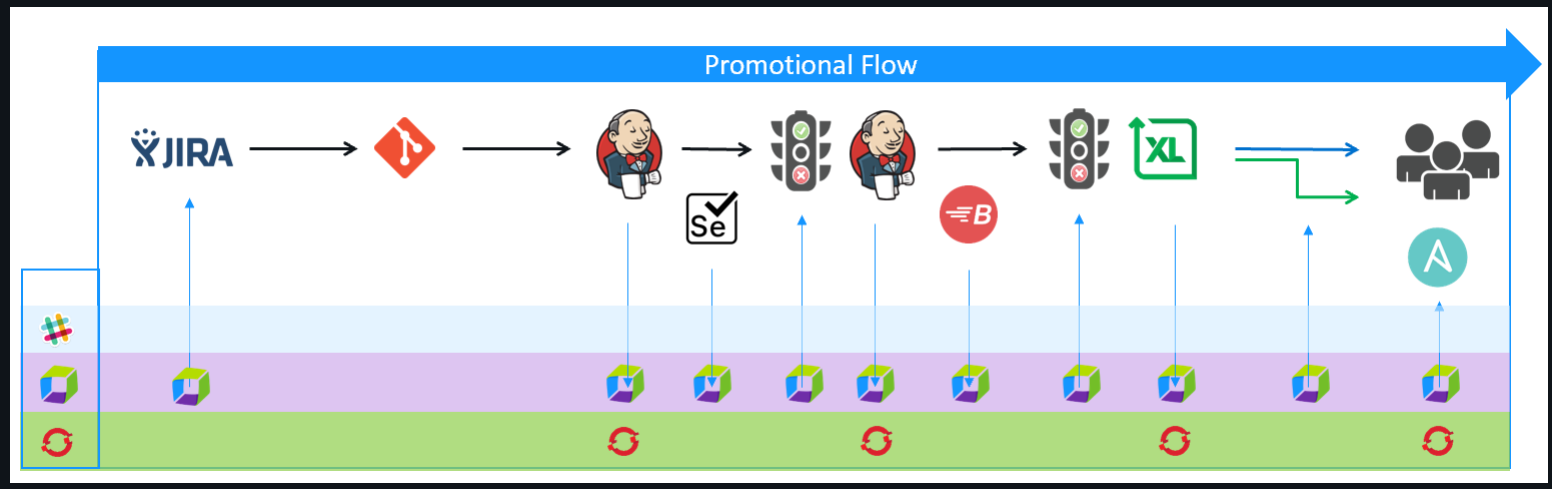


AIOps / ACM

Innovatively combining Agile, DevOps and cloud with AI to blaze a trail towards AIOps / autonomous cloud management (ACM)

Mick Miller, Sr DevOps Architect

KeyBank



Better

< 10 m

Feedback on feature branch commit

Faster

80%

Problem reduction

More Frequently

Daily

Release of authentication

Auto Regression Detection

Automating Quality Gates

Auto Release Feedback Loops

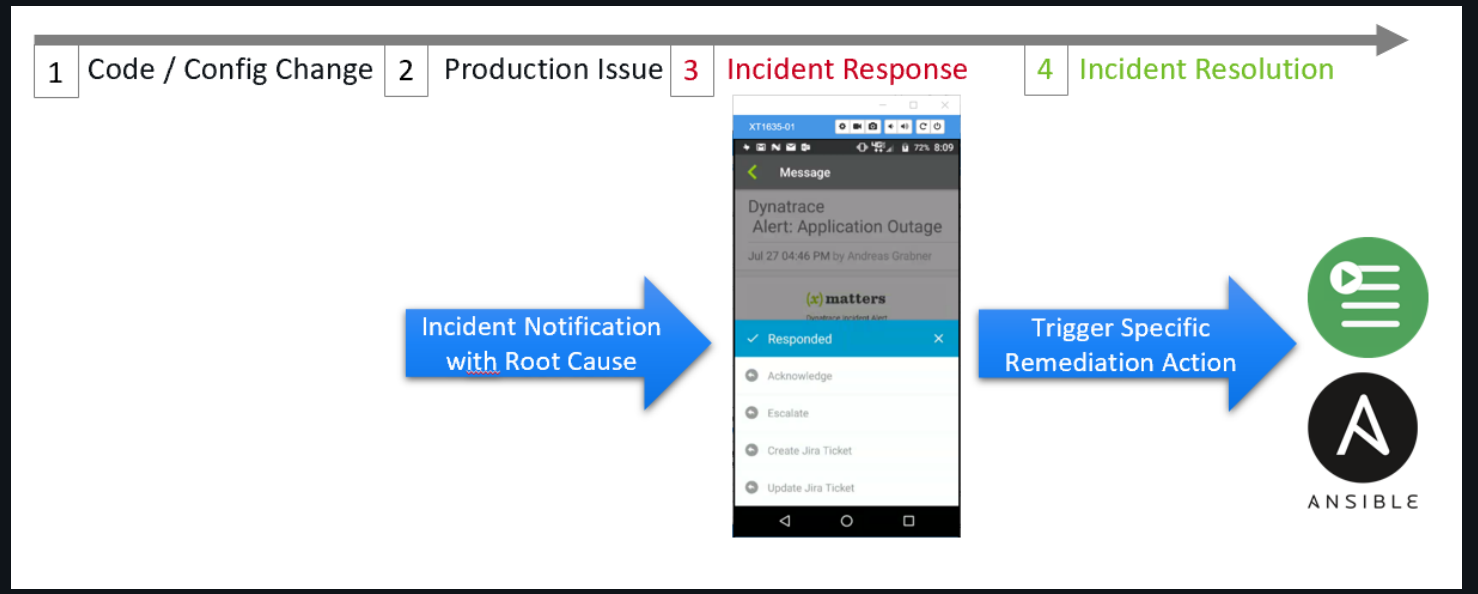
Automate Quality

Automate Deployment

Automate Operations

Auto-remediation

Stepping up automation maturity scale; integrating powerful AI into service management tools for predictive, self-healing operations

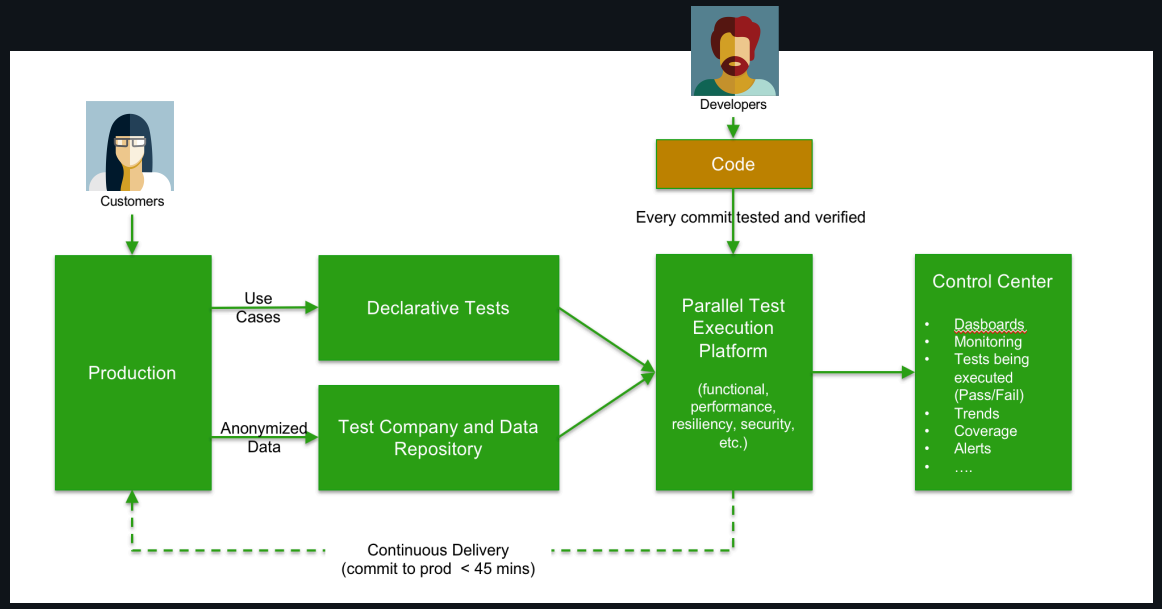


Jonathan Hayes
Vice President Global IT Service Excellence

- Building resilience – making better, resilient systems
- Smart auto-remediation speeds up MTTR reduces risk
- Prevent in CI/CD versus REPAIR in production with quality gates
- Auto-remediation-as-code (resilience in the code pack) address root cause and not just symptom

Performance-engineering-as-a-self-service

Integrating performance testing tools and embedding performance diagnostics into dev work flow for fast, automated feedback.



Sumit Nagal
Principal Engineer Quality

- Increased developer productivity
- Instant performance feedback on developer machines
- PR pipeline block code for check-in
- Regression detection on every deployment

keptn.sh - OpenSource framework for unbreakable pipeline and more …

Enterprise-grade framework for shipping and running cloud-native applications

Deployable on any Kubernetes cluster, keptn converts any Kubernetes cluster into a self-healing, autonomous cloud fabric.

keptn

CORE CAPABILITIES

- Automated multi-stage unbreakable delivery pipelines
- Self-healing blue / green deployments

- Event-driven runbook automation

DESIGN PRINCIPALS

- GitOps-based collaboration
- Operator patterns for all logic components

- Monitoring-and-operations-as-code

- Built-on and for Kubernetes

- Event-driven and serverless

- Pluggable tooling

Stay engaged with us!

1. Download a free copy of the Gartner MQ
   http://bit.ly/2FjBzcC
2. Take our survey https://dynatrace.ai/acsurvey
3. Catch our PurePerformance Podcast Series
   https://www.spreaker.com/user/pureperformance
4. Check out our Performance Clinics https://bit.ly/2pzXXIK
5. Jenkins Performance Signature
   https://www.dynatrace.com/news/blog/shift-left-in-jenkins-how-to-implement-performance-signature-with-dynatrace/
6. Check out the keptn ! – http://keptn.sh

How to cleanup

What to remove

- AWS
  - In your bastion host, run these commands
    - cd ~/terraform
    - Terraform destroy ← this will prompt for confirmation and takes 10+ minutes. May have to repeat
  - In AWS console
    - Verify EKS, VPC, and EC2 workers are deleted. If not, you may have to delete manually
    - Terminate the EC2 bastion host
- Dynatrace – Trial will expire in 15 days, but can remove/disable these:
  - Deploy Dynatrace → PaaS Integration → Lab Token
  - Settings → Integration → API Token → Lab Token
  - Settings → Integration → Problem notification → Self Healing function