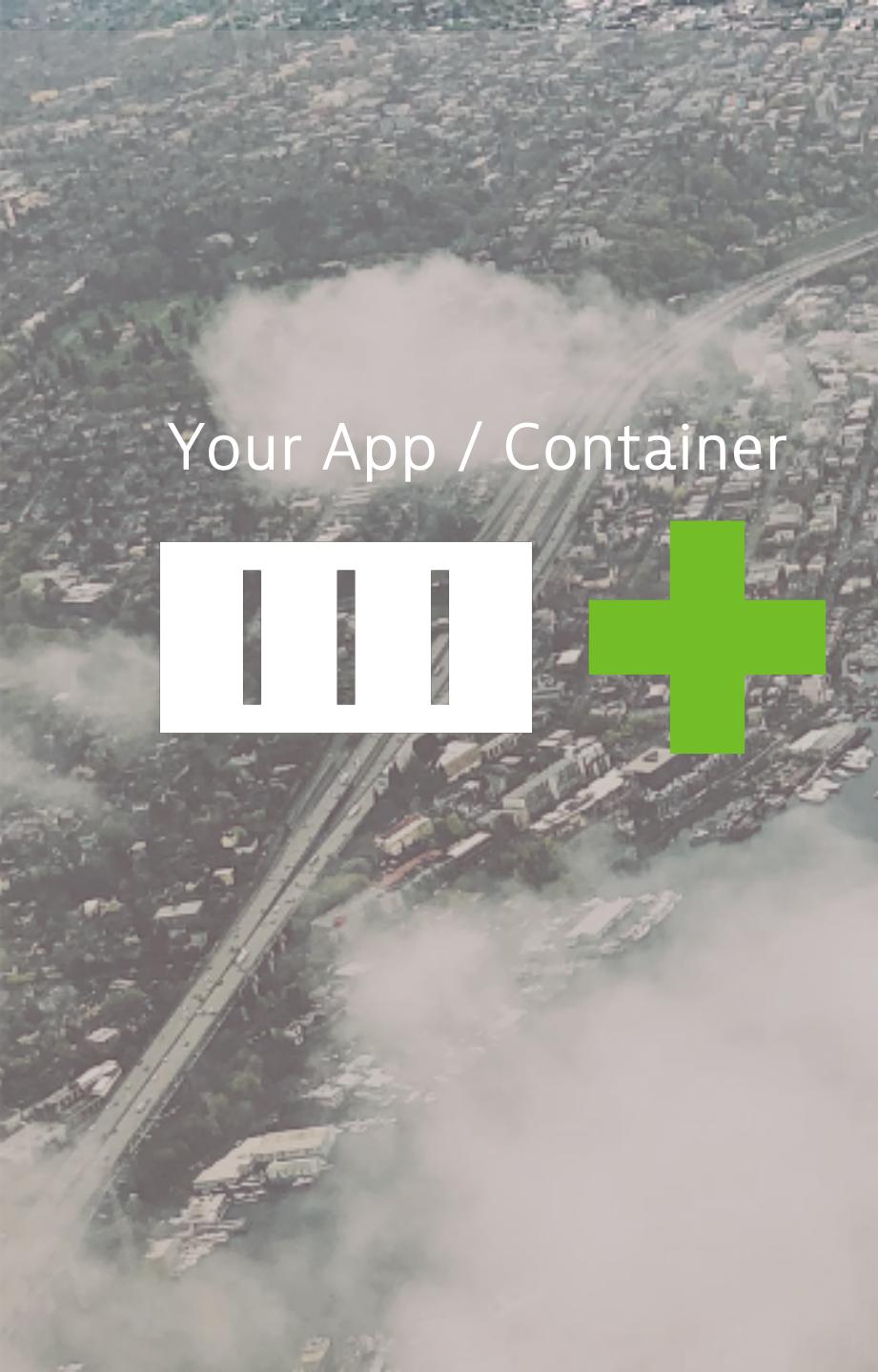


Dynatrace aws Hands-On Workshop

Unbreakable Continuous Delivery





Your App / Container



Pivotal **Cloud Foundry**®



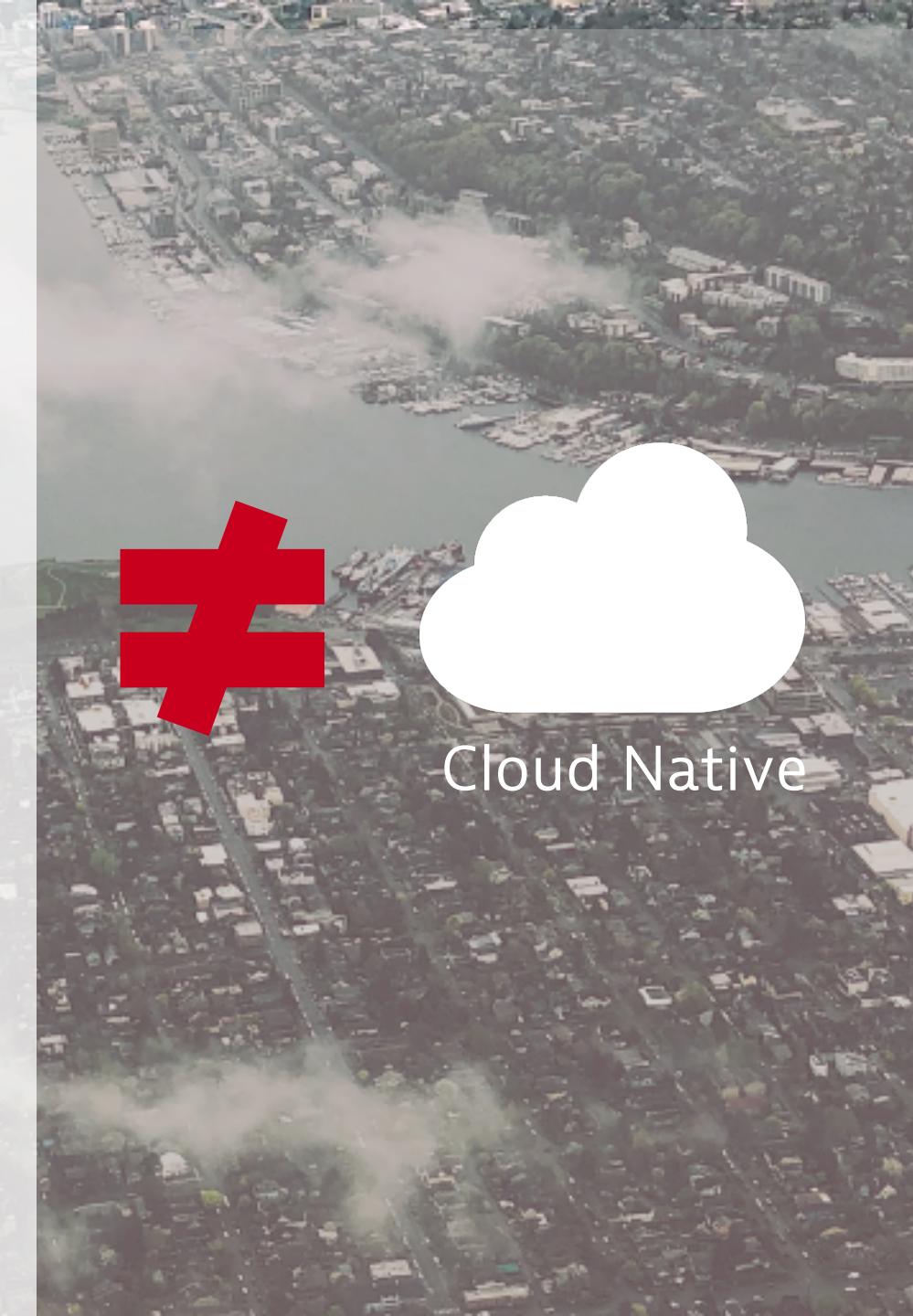
OPENSHIFT



kubernetes



Google Cloud Platform



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

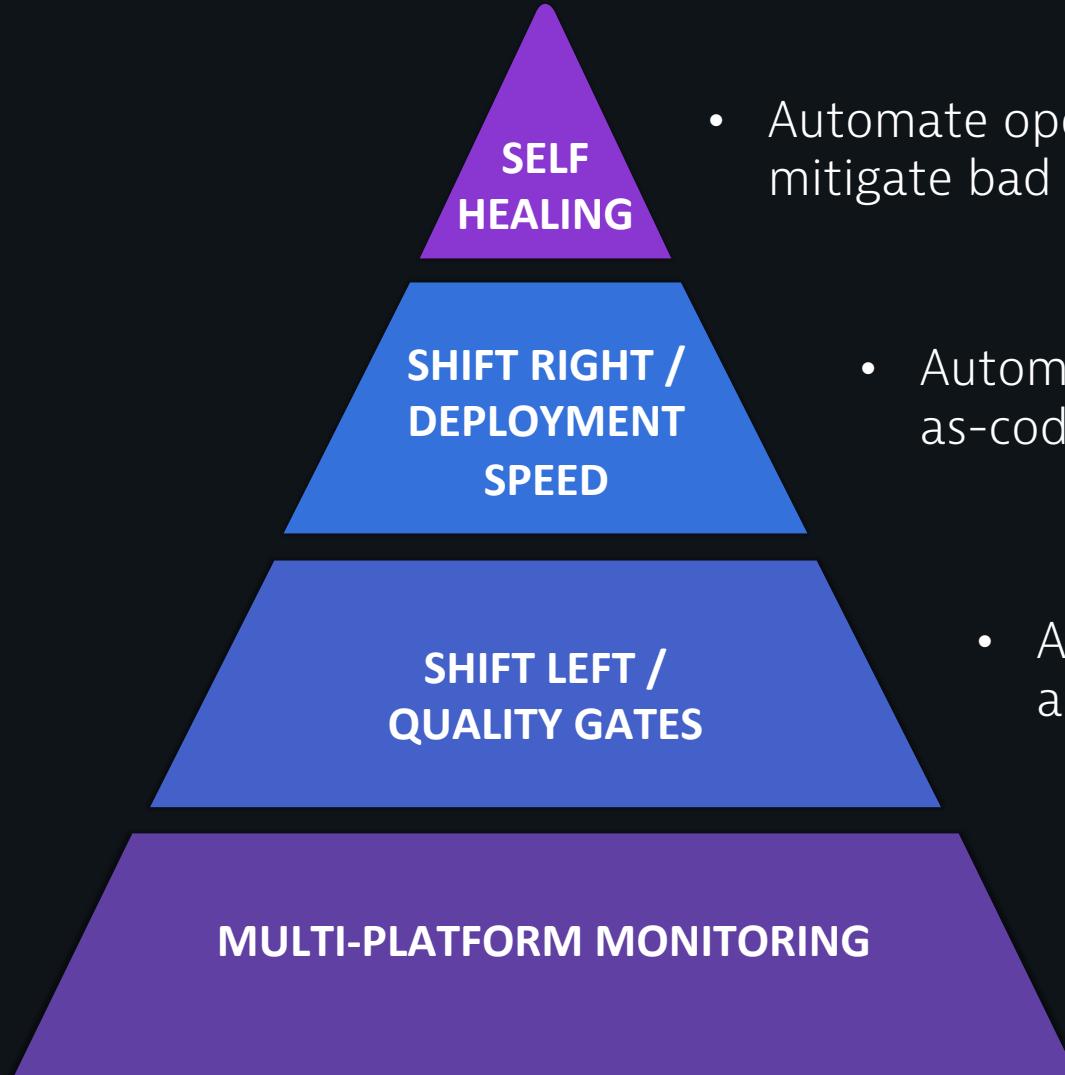
Code to Production
(Commit Cycle Time)

Business Impacting
Deployments

Per Production
Deployment

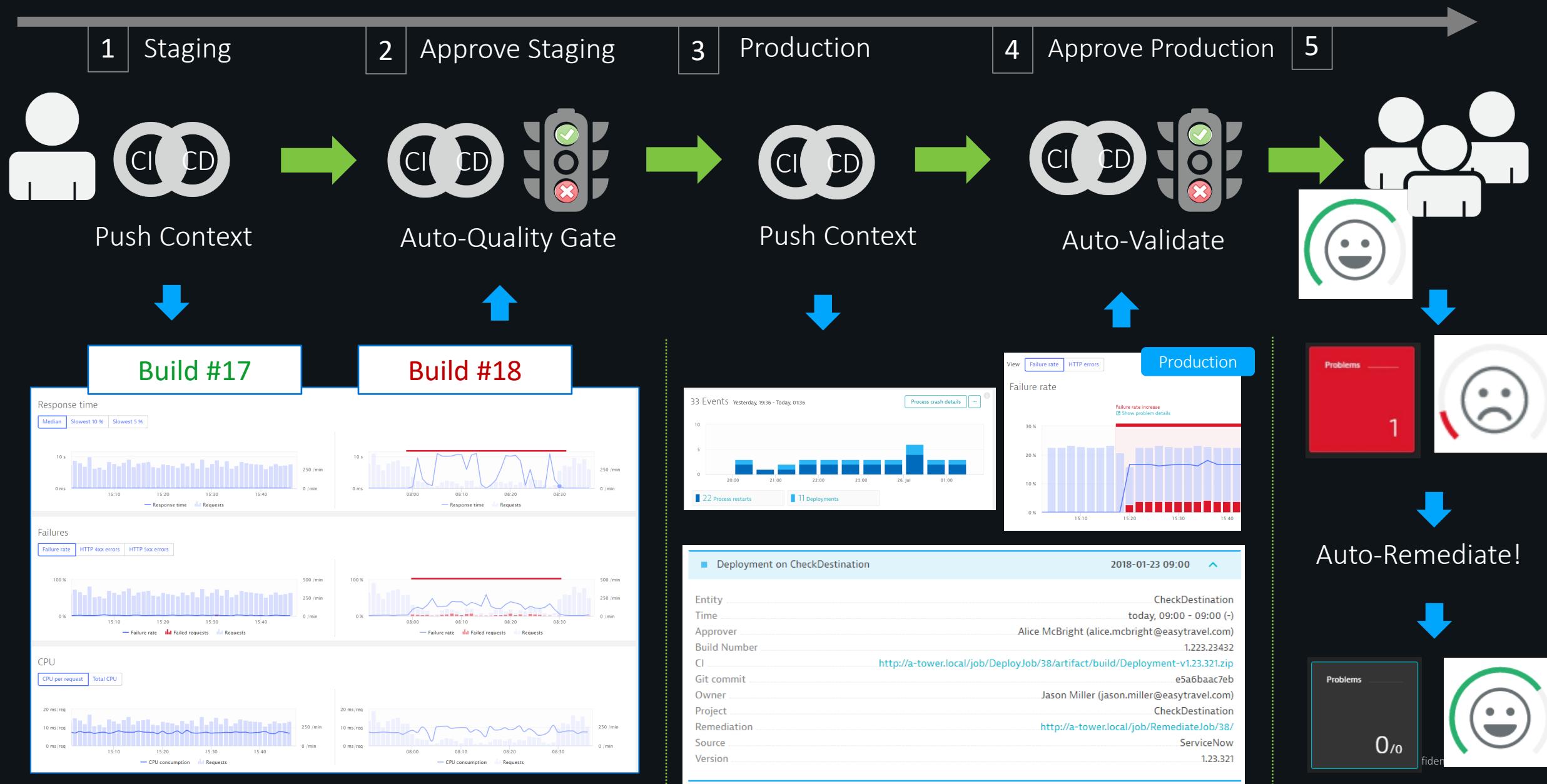
MTTR
(Mean Time to Repair)

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



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...

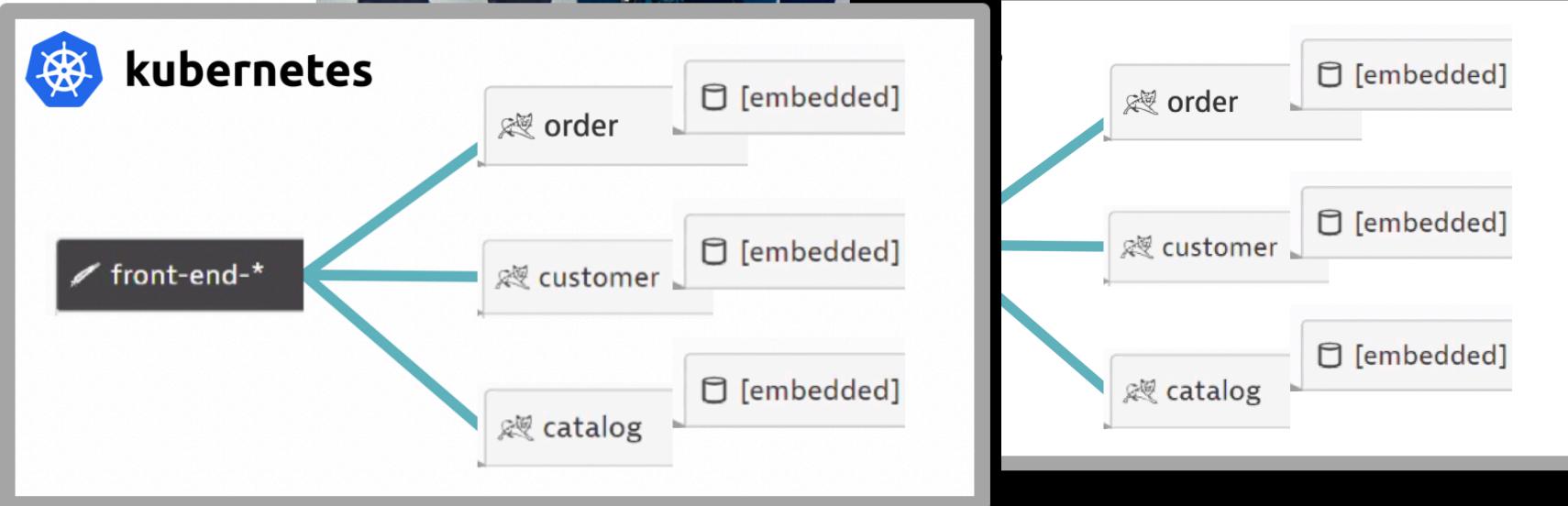
Workshop Application

- Order processing application
- Web UI with 3 java services with embedded database
- Components pre-compiled image, and staged on Kubernetes
- Application is deployed using deployment and service

Dynatrace
Order
Processing



Customer
Catalog
Search Items
Order





Jenkins Pipelines to deploy and run performance tests

Screenshot of the Jenkins dashboard showing pipelines and build executor status.

Pipeline List:

All	S	W	Name ↓
		deploy-production	
		deploy-service	
		deploy-staging	
		load-test	

Icon Legend: S M L

Build Executor Status:

Build Queue	Build Executor Status
No builds in the queue.	1 Idle 2 Idle

A large blue arrow points from the bottom left towards the pipeline table.

Pipeline Details:

Declarative: Checkout SCM	Deploy	Health Check	Push Dynatrace Deployment Event	Run e2e load test	quality gate
859ms	23s	1min 2s	20s	5min 49s	2s



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



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 1 Affected infrastructure 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 Impacted users

270k 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 69 /min Service method All dynamic requests

Failure rate increase

by a failure rate increase to 9.86 %

Affected requests 69 /min Service method All dynamic requests

Root cause

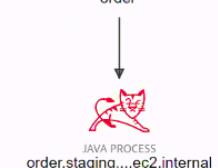
Based on our dependency analysis all incidents have the same root cause

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 down

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

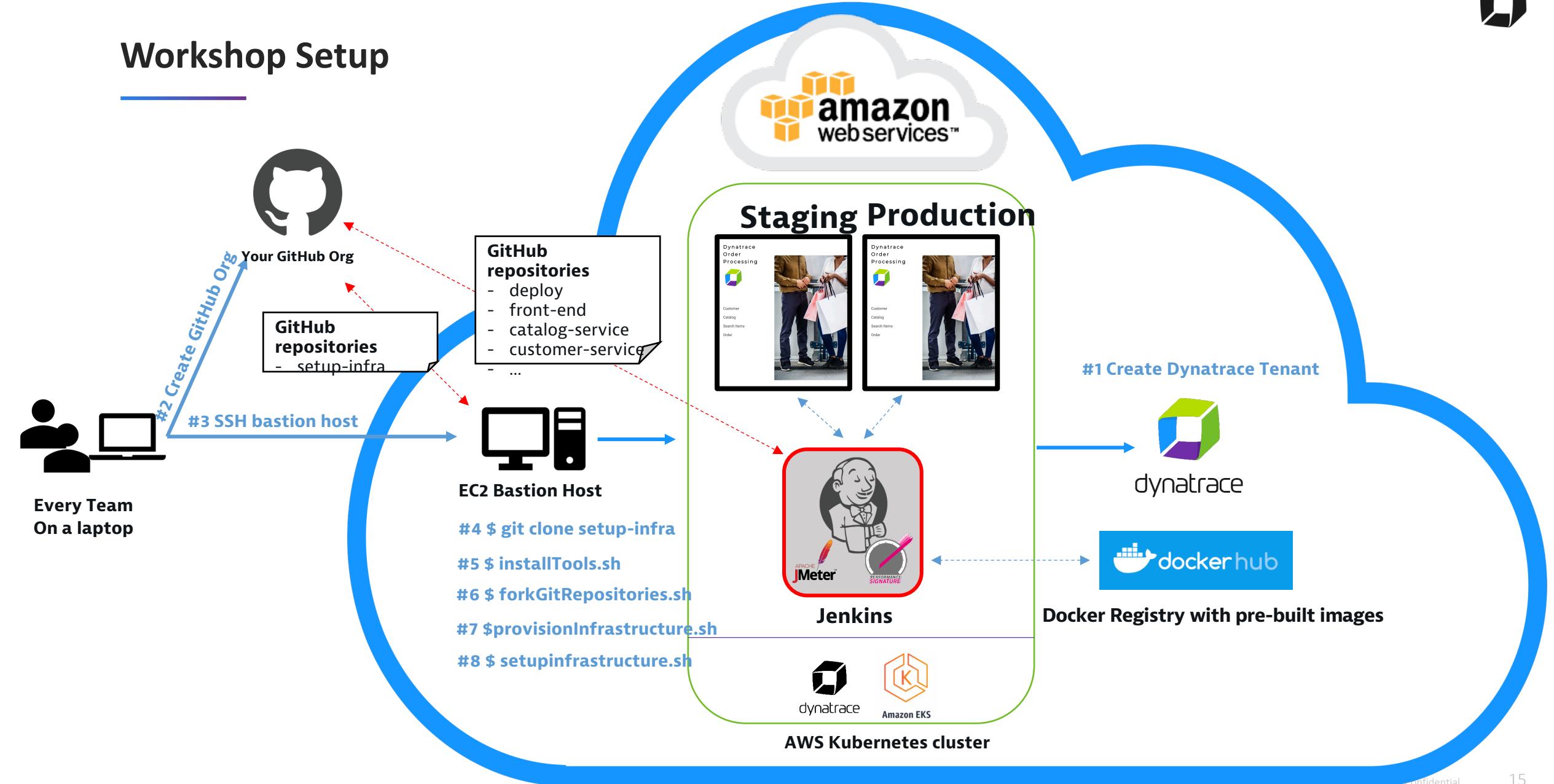
- *Deploy the application*
- *Explore & Finish Dynatrace Configuration*
- *Quality Gate*
- *Self healing*

4. Recap

Demo infrastructure setup

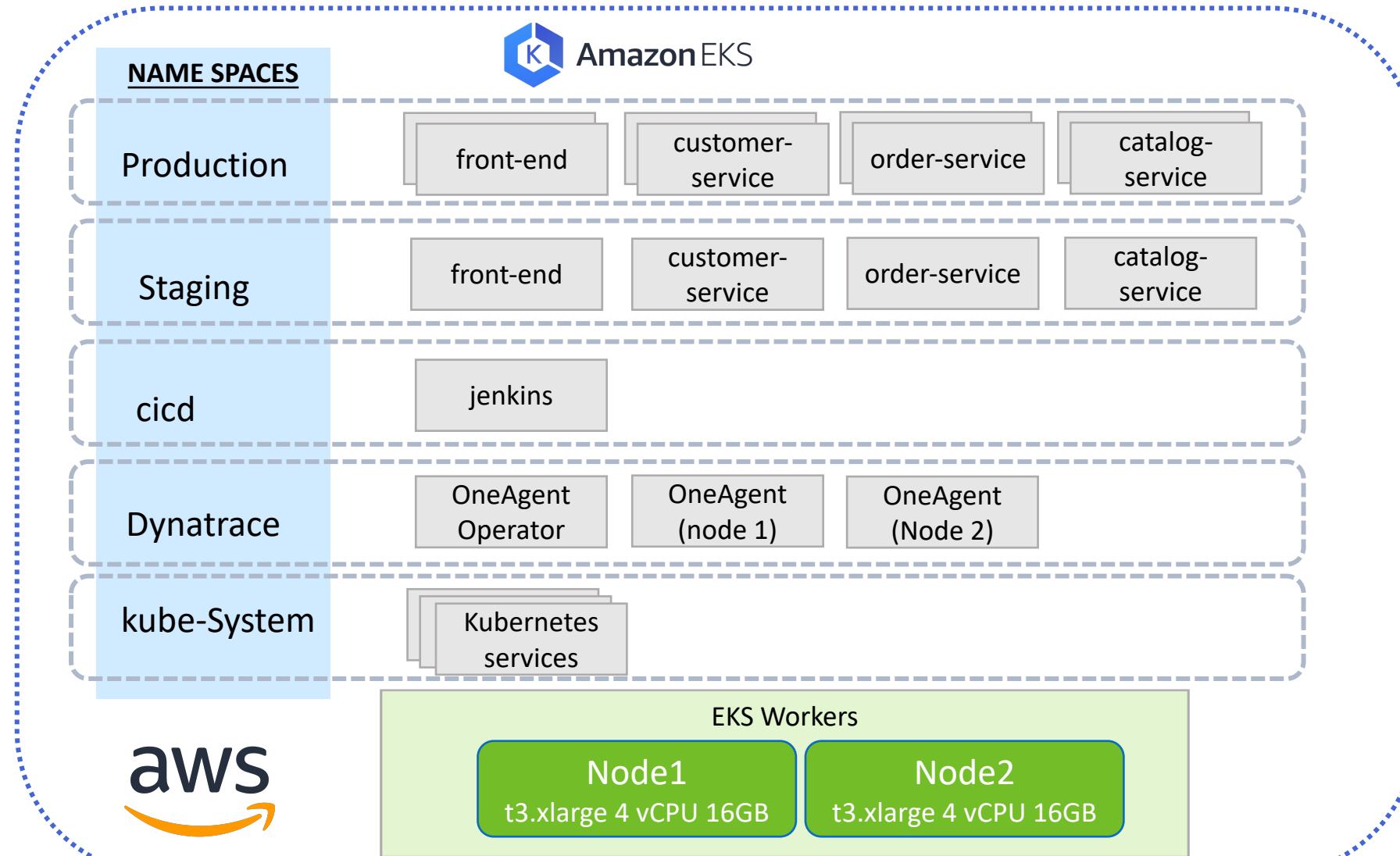


Workshop Setup





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

```
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: {}
```

The screenshot shows the Dockerhub interface. At the top, there's a header with navigation icons, a URL bar showing https://hub.docker.com, and user information. Below the header, a banner for 'dockercon19' is visible. The main area displays a search bar with 'Search for great content (e.g., mysql)' and a dropdown menu set to 'robjahn'. A large blue arrow labeled '#2' points to the first repository card. The cards are arranged vertically and show the following details:

- robjahn / **customer-service** Updated 3 days ago
- robjahn / **order-service** Updated 3 days ago
- robjahn / **front-end** Updated 5 days ago
- robjahn / **catalog-service** Updated 5 days ago



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

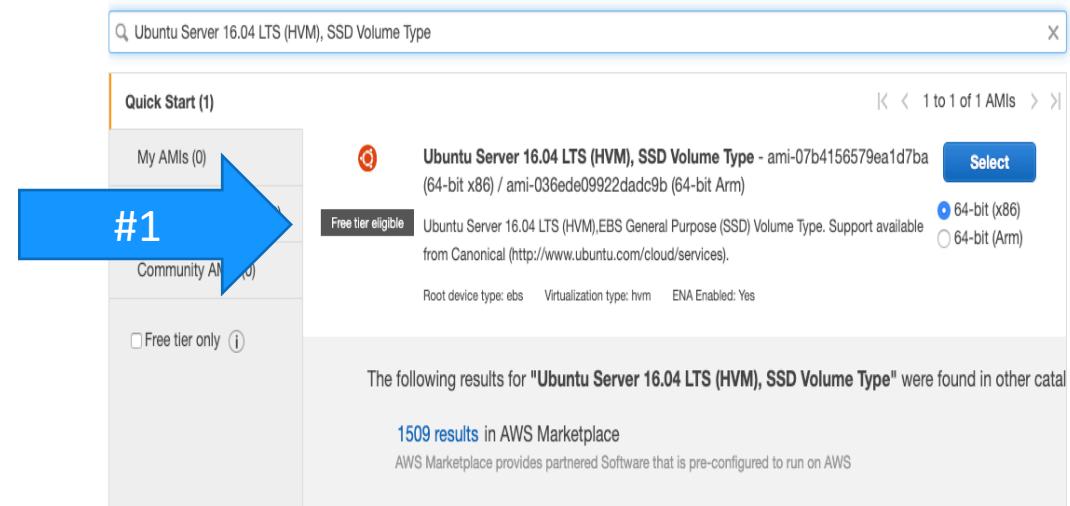
Demo infrastructure setup

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.micro
 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

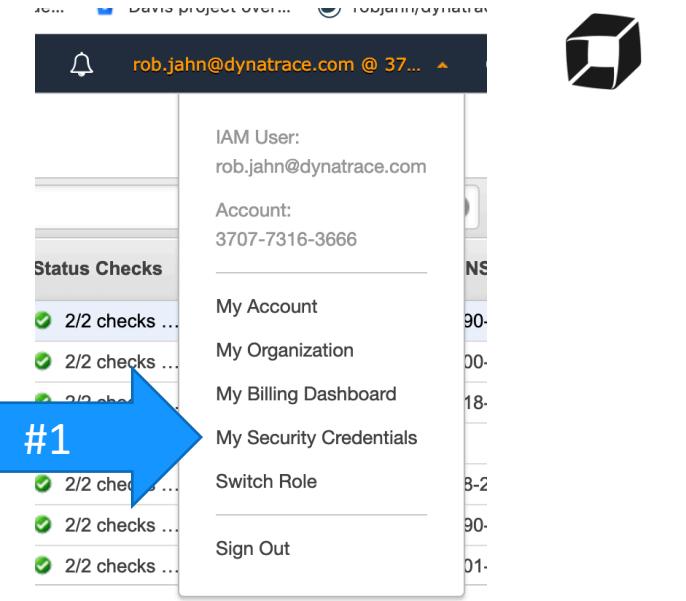


Create Bastion via aws cli

- 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>
 - CLUSTER_REGION=<example us-west-2>
 - RESOURCE_PREFIX=<example your last name>
- Run script
 - ./bastion.sh

The screenshot shows the AWS Lambda console interface. At the top, there are three buttons: 'Create Key Pair' (highlighted in blue), 'Import Key Pair', and 'Delete'. Below these buttons is a search bar with the placeholder text 'Filter by attributes or search by keyword'. The main area displays a table of key pairs. The columns are 'Key pair name' and 'Fingerprint'. The table contains four rows:

Key pair name	Fingerprint
Ansible	de:b3:3f:f6:99:e0:c3:56:5b:87:2a
DevOps	7a:4b:a4:76:3b:a2:58:87:d2:c5:d
serviceWorkshop	a0:6a:50:85:80:96:af:ea:f3:8e:da
ssh	59:68:5a:20:1a:94:c5:f5:f3:2d:e9



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 than 'us-east-1' then you will have to 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

```
[ec2-user@ip-172-31-30-98 ~]$ aws configure
AWS Access Key ID [*****4JYA]:
AWS Secret Access Key [*****zSDe]:
Default region name us-east-1
Default output format [None]: json #2

[ec2-user@ip-172-31-30-98 ~]$ aws ec2 describe-instances
{
  "Reservations": [
    {
      "Instances": [
        {
          "Monitoring": {
            "State": "disabled"
          },
          "PublicDnsName": "ec2-52-38-227-172.",
          "StateReason": {
            "Message": "",
            "Code": ""
          },
          "State": {
            "Code": 16,
            "Name": "running"
          },
          "EbsOptimized": false,
          "LaunchTime": "2018-03-05T09:25:38.0 #3
          ...
        }
      ]
    }
  ]
}
```



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

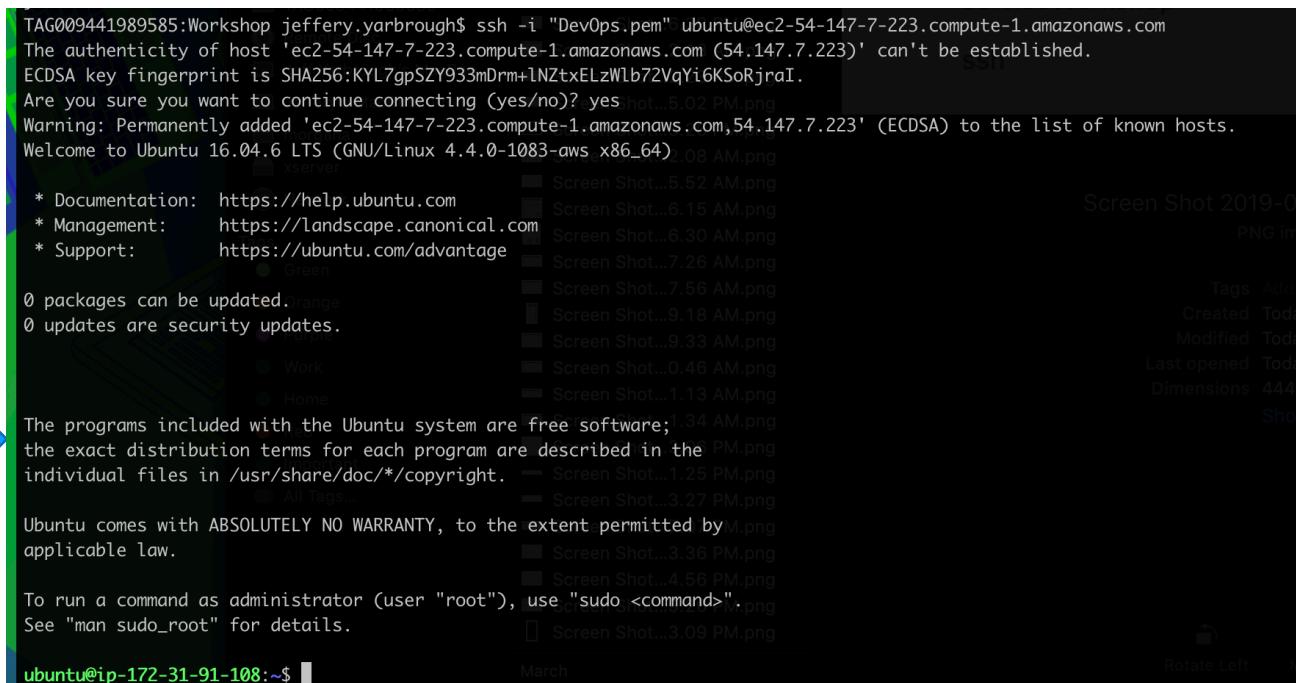
The screenshot shows the AWS EC2 Instances page. At the top, there is a search bar with the text "search : jahn-demo-app-bastion" and a "Connect" button. Below the search bar is a table with columns: Name, Instance ID, Instance Type, Availability Zone, and Instance State. The first row in the table is highlighted with a blue background and contains the following information: Name: "jahn-demo-app-bastion", Instance ID: "i-00e51837f9353c525", Instance Type: "t2.micro", Availability Zone: "us-east-1a", and Instance State: "running". Below the table, the instance details are summarized: Instance: "i-00e51837f9353c525 (jahn-demo-app-bastion)" and Public DNS: "ec2-3-89-140-245.compute-1.amazonaws.com". There are tabs for Description, Status Checks, Monitoring, and Tags, with Tags being the active tab. Below the tabs is a "Add/Edit Tags" button. A table titled "Tags" shows two entries: "Name" with value "jahn-demo-app-bastion" and "Owner" with value "jahn".



Connect to EC2 instance

1. Connect to host. For example in Mac @ the terminal in folder with pem file
 - ssh -i "DevOps.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



The screenshot shows a terminal window with the following text:

```
TAG009441989585:Workshop jeffery.yarbrough$ ssh -i "DevOps.pem" ubuntu@ec2-54-147-7-223.compute-1.amazonaws.com
The authenticity of host 'ec2-54-147-7-223.compute-1.amazonaws.com (54.147.7.223)' can't be established.
ECDSA key fingerprint is SHA256:KYL7gpSZY933mDrm+lNztxELzWlb72VqYi6KSoRjraI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-147-7-223.compute-1.amazonaws.com,54.147.7.223' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1083-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.

ubuntu@ip-172-31-91-108:~$
```

Rotate Left

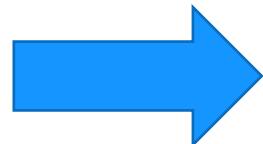
Confidential



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



```
ubuntu@ip-172-31-91-108:~/setup-infra$ aws configure
AWS Access Key ID [None]: AKIAVMGPJPNXGESVHZWK
AWS Secret Access Key [None]: MlbY2uvsdFpGHqR4g00cEZRh+04CeSZMuUBRKbX
Default region name [None]: us-east-1
Default output format [None]: json
```



Install Prerequisites Tools

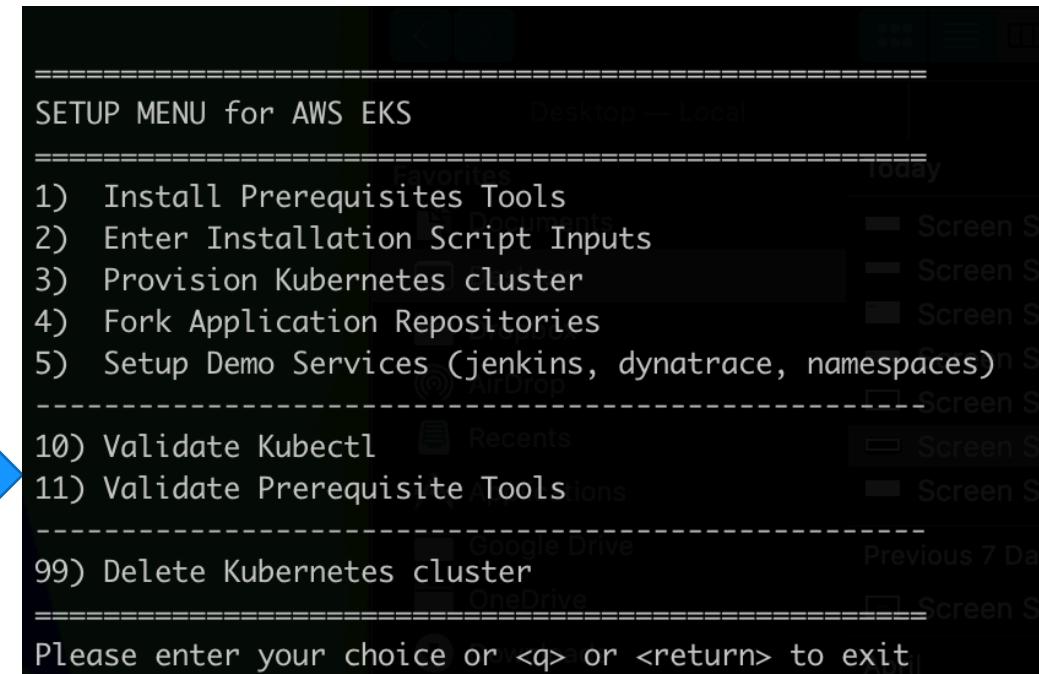
1. Clone GitHub repo

- git clone <https://github.com/dt-kube-demo/setup-infra.git>
- cd setup-infra

```
ubuntu@ip-172-31-91-108:~$ git clone https://github.com/dt-kube-demo/setup-infra.git
Cloning into 'setup-infra'...
remote: Enumerating objects: 50, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 329 (delta 14), reused 24 (delta 8), pack-reused 279
Receiving objects: 100% (329/329), 2.45 MiB | 0 bytes/s, done.
Resolving deltas: 100% (191/191), done.
Checking connectivity... done.
```

2. This will install unix tools such as kubectl, jq, etc...

- Run
 - ./setup.sh <deployment type> (./setup.sh eks)
 - eks = AWS
 - aks = Azure
 - Select Option 1



Demo infrastructure setup

Provision Infrastructure



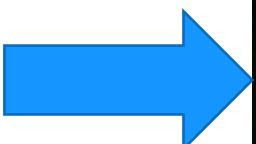
Prepare bastion host – Enter Installation Script Inputs

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’



```
Please enter the values for provider: AWS EKS
Press <enter> to keep the current value
=====
Dynatrace Tenant ID (e.g. abc12345)      (current: DYNATRACE_TENANT_PLACEHOLDER)
Dynatrace Host Name                         (current: DYNATRACE_HOSTNAME_PLACEHOLDER)
Dynatrace API Token                         (current: DYNATRACE_API_TOKEN_PLACEHOLDER)
Dynatrace PaaS Token                        (current: DYNATRACE_PAAS_TOKEN_PLACEHOLDER)
GitHub User Name                           (current: GITHUB_USER_NAME_PLACEHOLDER)
GitHub Personal Access Token               (current: PERSONAL_ACCESS_TOKEN_PLACEHOLDER)
GitHub User Email                          (current: GITHUB_USER_EMAIL_PLACEHOLDER)
GitHub Organization                       (current: GITHUB_ORG_PLACEHOLDER) : jyarbrough
PaaS Resource Prefix (e.g. lastname)       (current: RESOURCE_PREFIX_PLACEHOLDER)
Cluster Region (e.g.us-east-1)              (current: CLUSTER_REGION_PLACEHOLDER) : us-east-1
=====

Please confirm all are correct:
=====
Dynatrace Tenant          iCloud    : ibg73613
Dynatrace Host Name        iCloud    : ibg73613.live.dynatrace.com
Dynatrace API Token        Locations : m3sTRYBnRSeaSbueCqh0F
Dynatrace PaaS Token       Locations : lojneX1rTx0kEJS1uJxEO
GitHub User Name           TAGS     : pcjeffmac
GitHub Personal Access Token TAGS     : 0c04a430bd5b36c0d9ba7a5d20ea7d1bcf6af477
GitHub User Email           Remote Dis : pcjeffmac@gmail.com
GitHub Organization         Jeff      : jyarbrough-workshop
PaaS Resource Prefix        Jeff      : jyarbrough
Cluster Region              morgan   : us-east-1
=====

Is this all correct? (y/n) :
```



Prepare bastion host – Provision Kubernetes Cluster

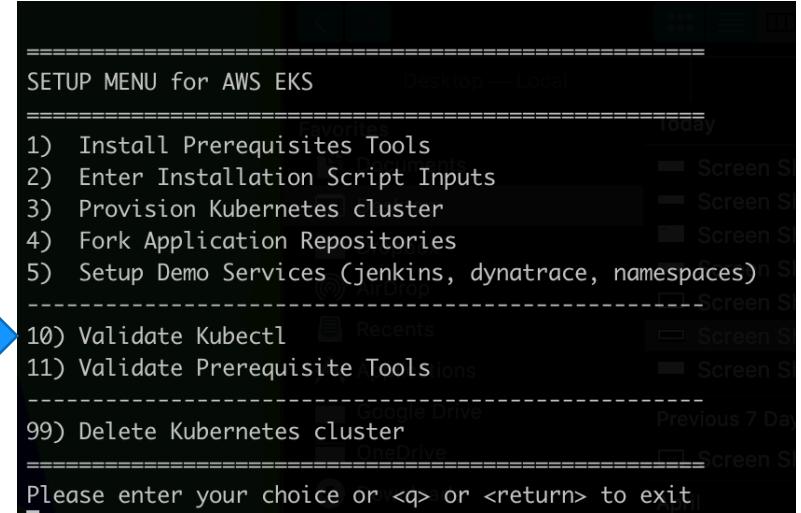
Back on the bastion host, to fork the demo app repos, run these commands:

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

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

Provision the Kubernetes Cluster

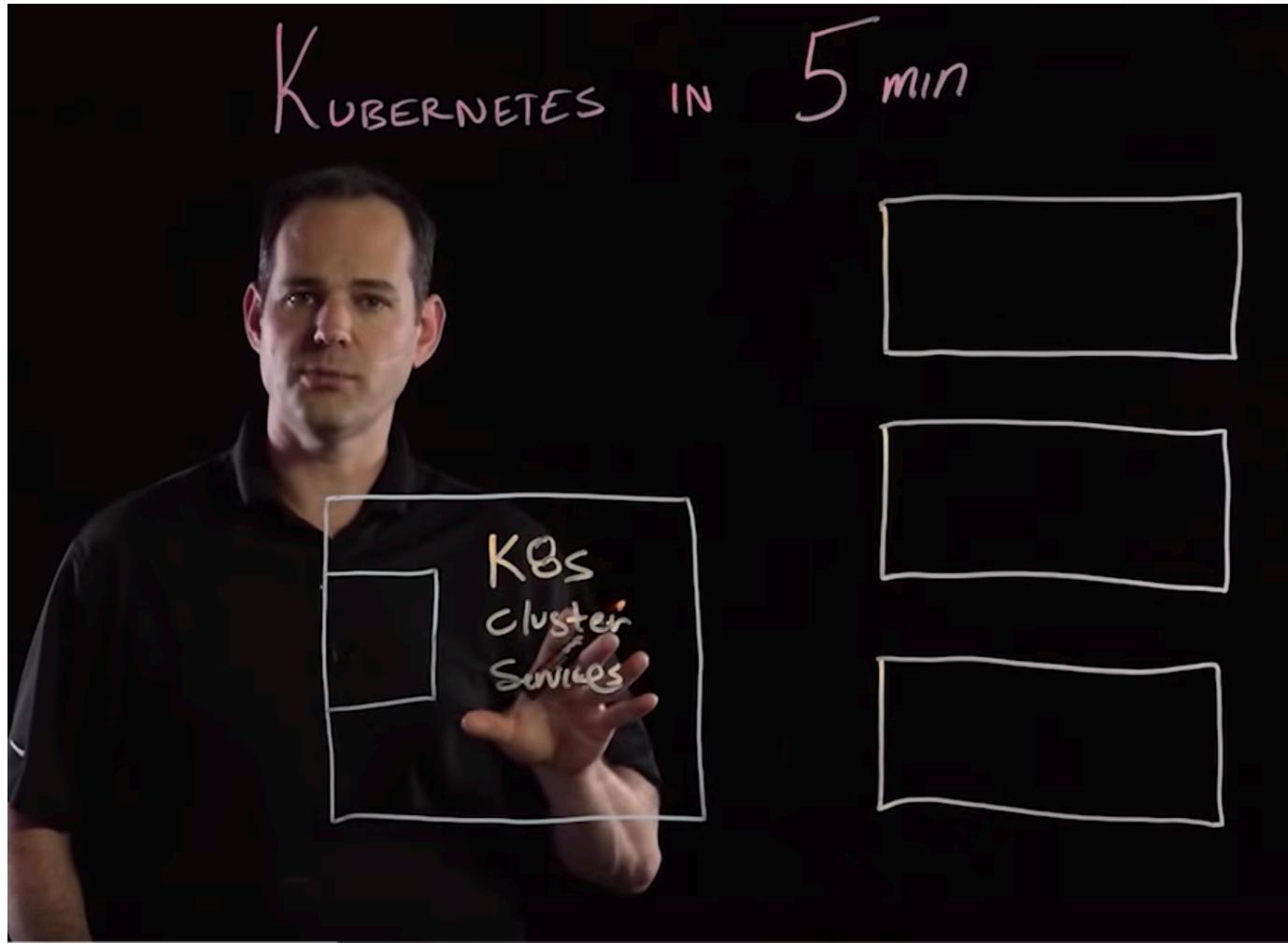
```
Creating AKS Cluster: jyarbrough-dt-kube-demo-cluster
[+] using region us-east-1
[+] setting availability zones to [us-east-1f us-east-1e]
[+] subnets for us-east-1f - public:192.168.0.0/19 private:192.168.64.0/19
[+] subnets for us-east-1e - public:192.168.32.0/19 private:192.168.96.0/19
[+] nodegroup "ng-2f3e6c59" will use "ami-0abcb9f9190e867ab" [AmazonLinux2/1.12]
[+] creating EKS cluster "jyarbrough-dt-kube-demo-cluster" in "us-east-1" region
[+] will create 2 separate CloudFormation stacks for cluster itself and the initial nodegroup
[+] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1'
[+] 2 sequential tasks: { create cluster control plane "jyarbrough-dt-kube-demo-cluster", create nodegroup "ng-2f3e6c59"
[+] building cluster stack "eksctl-jyarbrough-dt-kube-demo-cluster-cluster"
[+] deploying stack "eksctl-jyarbrough-dt-kube-demo-cluster-cluster"
```



While the cluster is building...

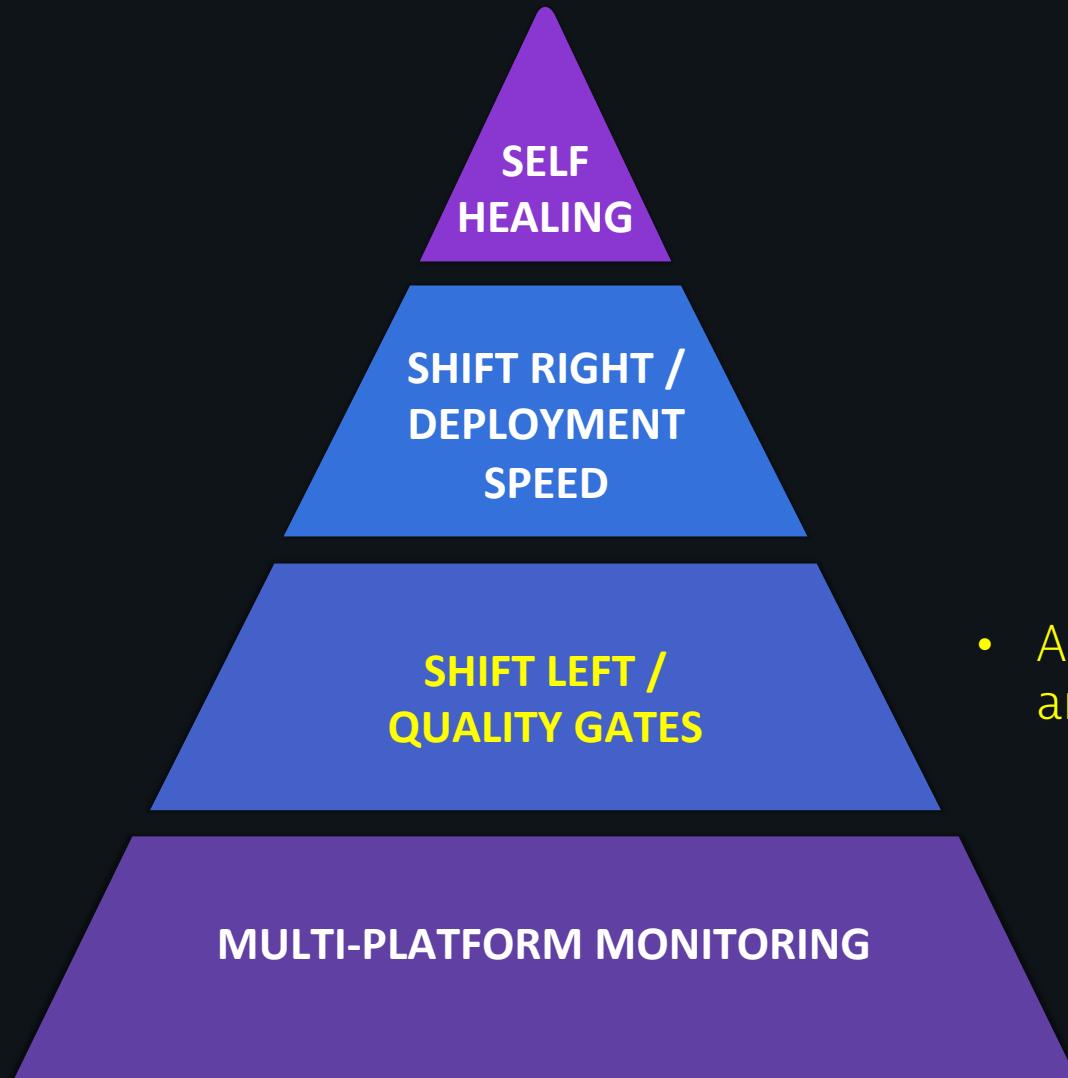


Kubernetes



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

Key blue print – unbreakable continuous delivery model

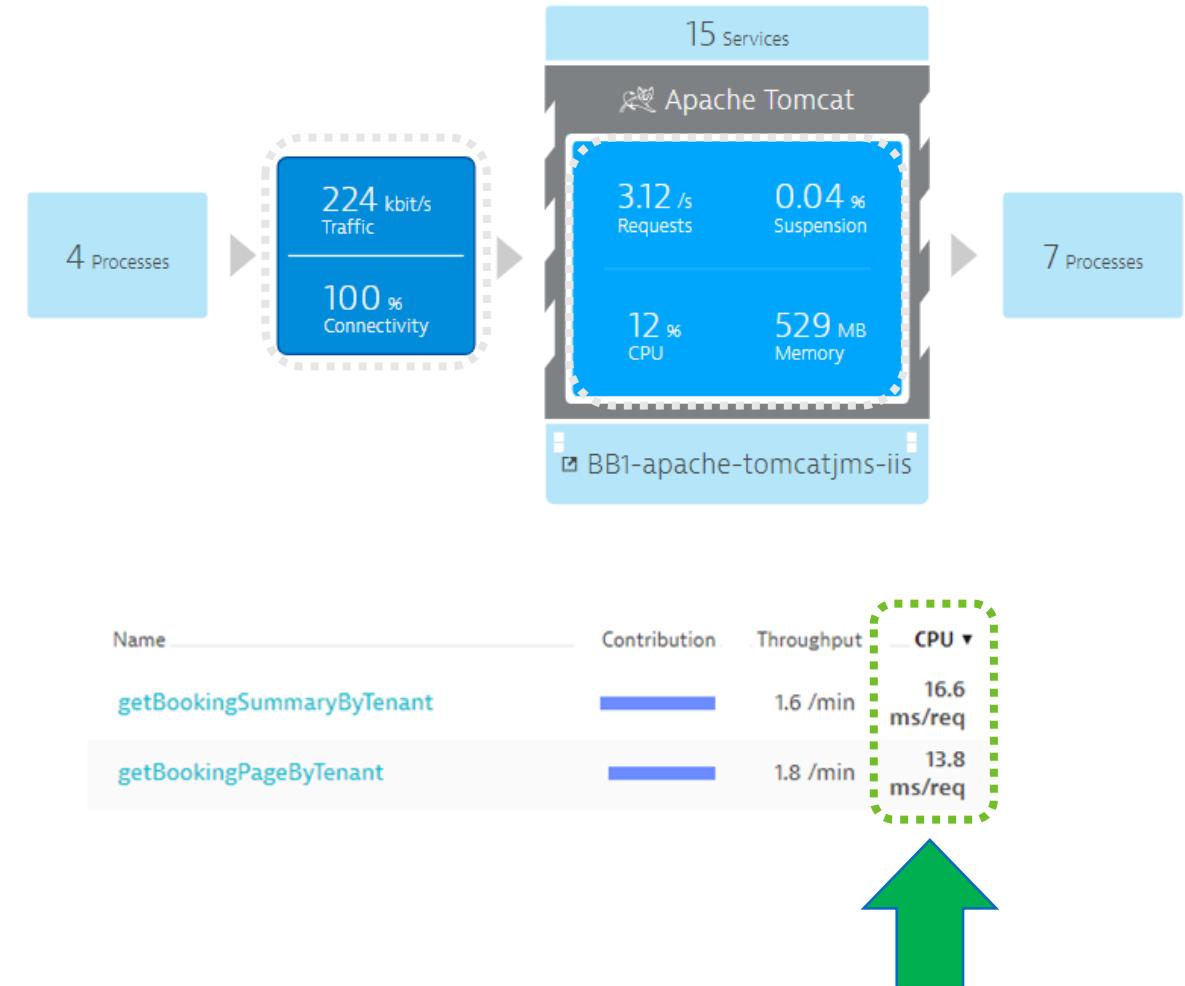


- Automate quality (shift-left) – automate the pipeline and stop bad code changes before they reach prod



Quality Gate: Detect Change in Resource Consumption

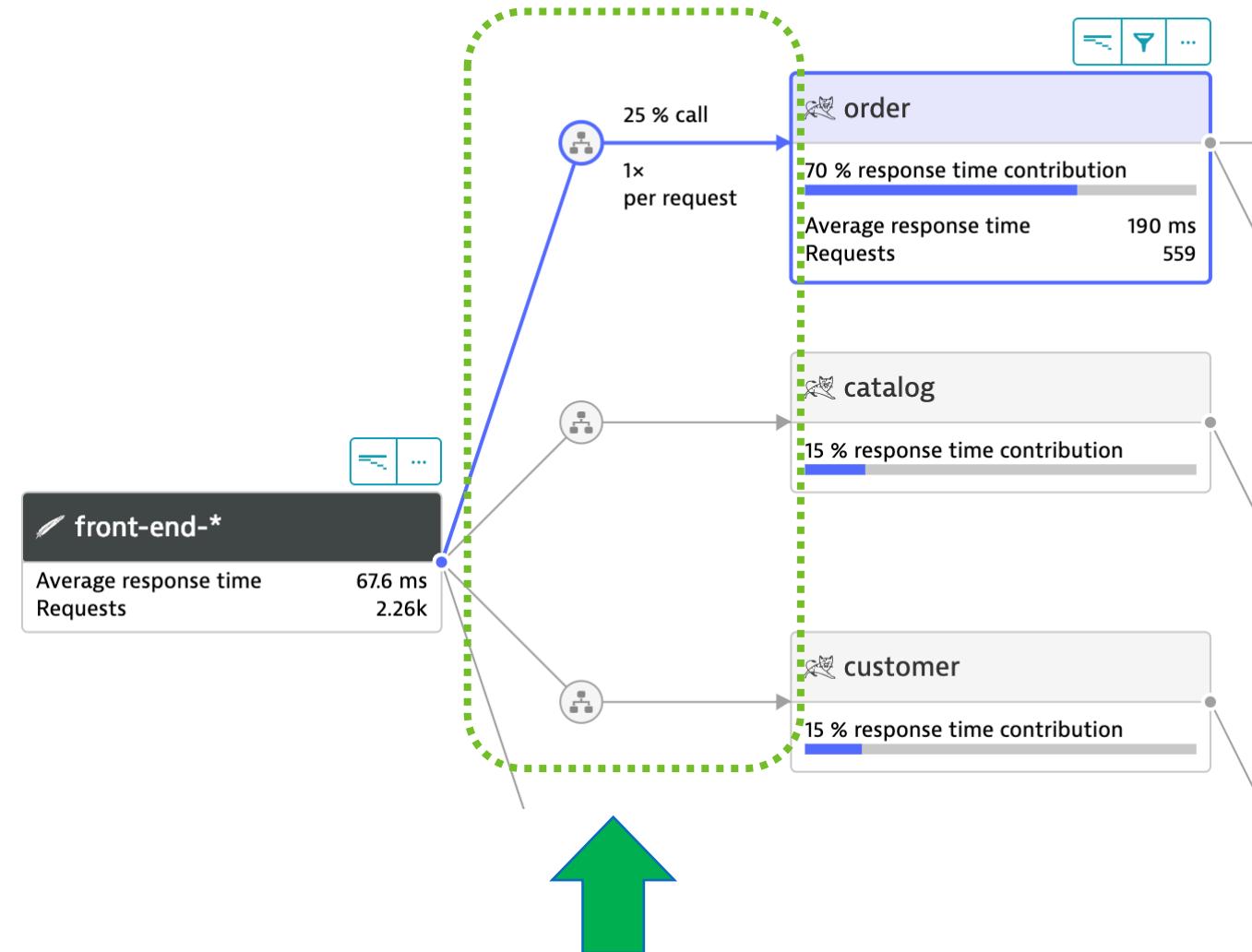
- Use Cases
 - Bad coding leads to higher costs?
- Metrics
 - Memory Usage
 - Bytes Sent/Received
 - Overall CPU
 - CPU per Transaction Type
- How to query?
 - Some through CloudWatch API
 - Dynatrace Timeseries API





Quality Gate: Detect Change in Dependencies

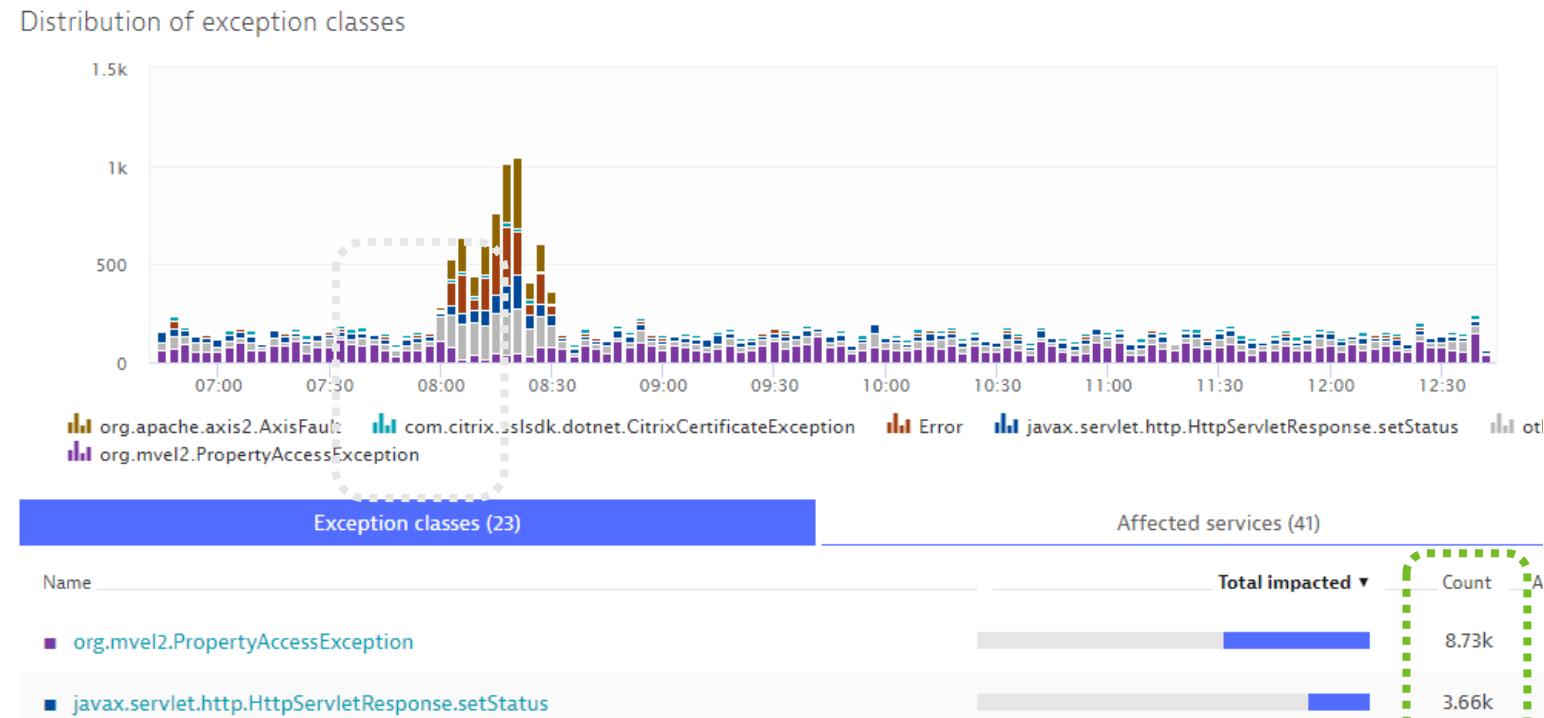
- Use Cases
 - Do we have new dependencies? On Purpose?
 - Are we connecting to the services we are supposed to connect?
 - How many container instances are required?
- Metrics
 - Number of incoming / outgoing dependencies
 - Number of instances running on
- How to query?
 - Maybe CloudWatch API
 - Dynatrace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



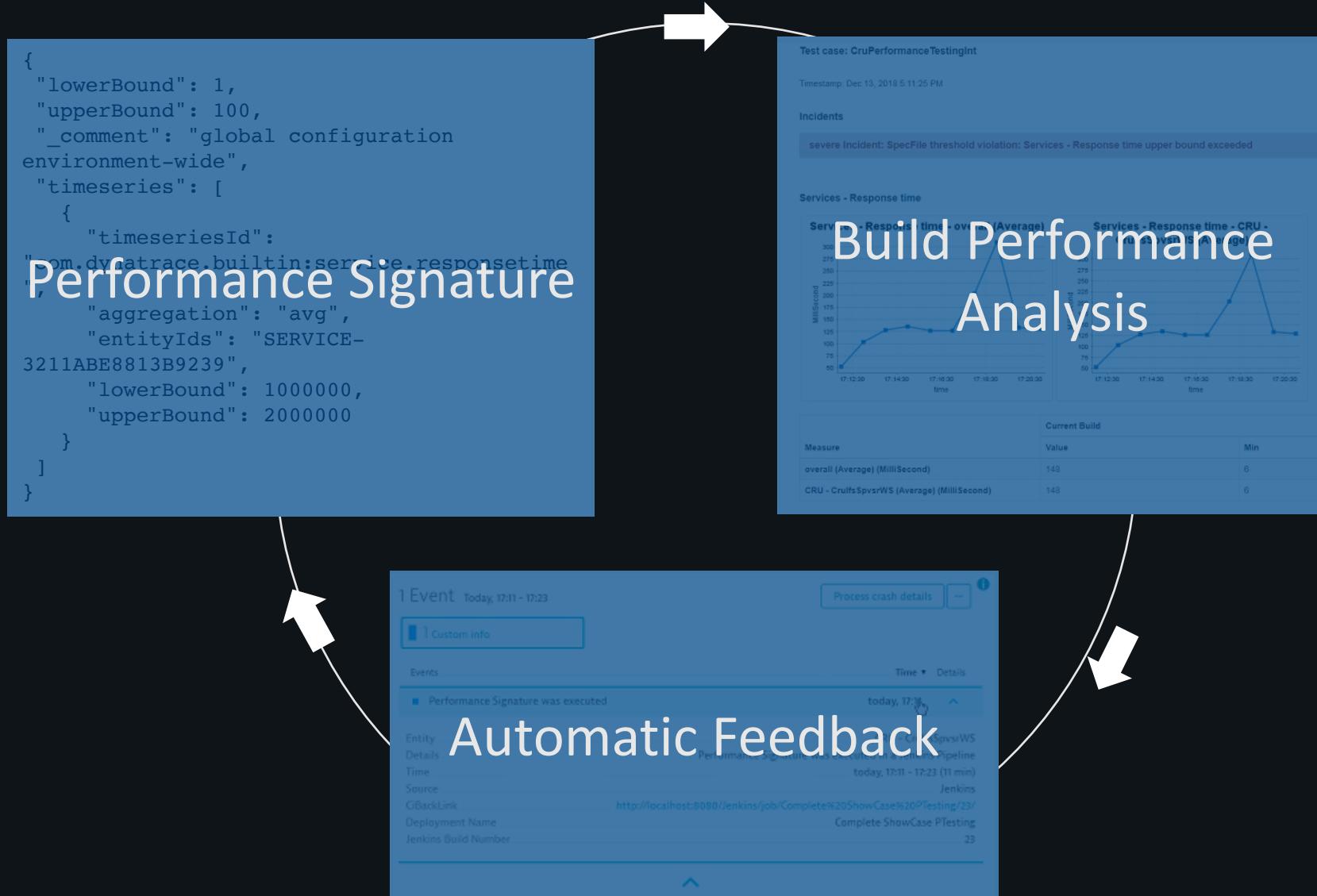


Quality Gate: Detect Change in Application Exception Handling

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

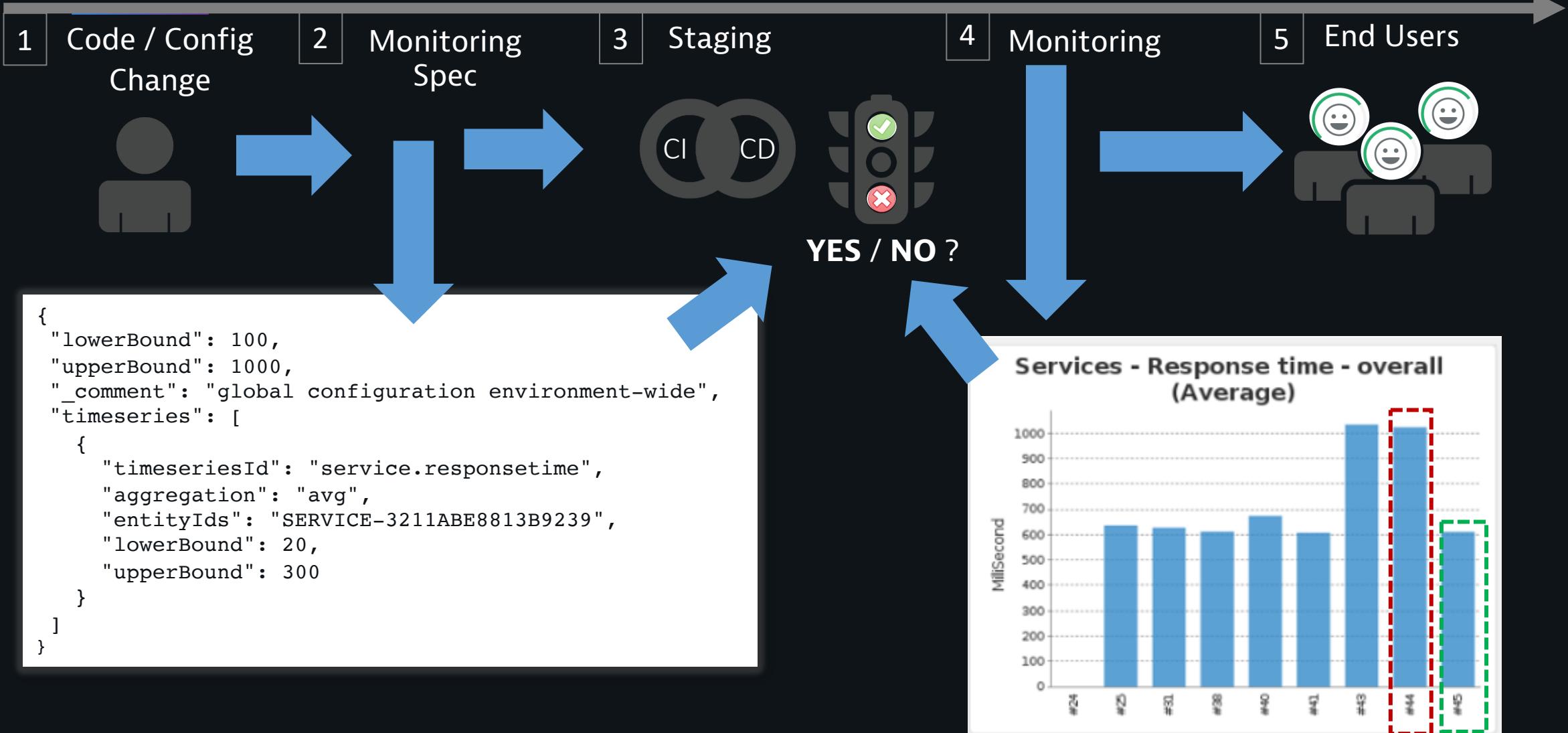


Shift-left quality gates – monitoring-spec-as-code





Integrate performance signature – monitoring-as-code



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

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.

Test Run 4

Test Run 5

2k
1.5k
1k
500
0

01:30 01:35 01:40 01:45 01:50 01:55 02:00 02:05 02:10 02:15 02:20 02:25 02:30 02:35 02:40

Distribution

- No interaction with services or queues
- Database usage
- Service execution

Top findings

- Active wait time +1.02 s
- Network IO time +67.5 ms



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?

Check 4

- Did we introduce new "hidden" exceptions?

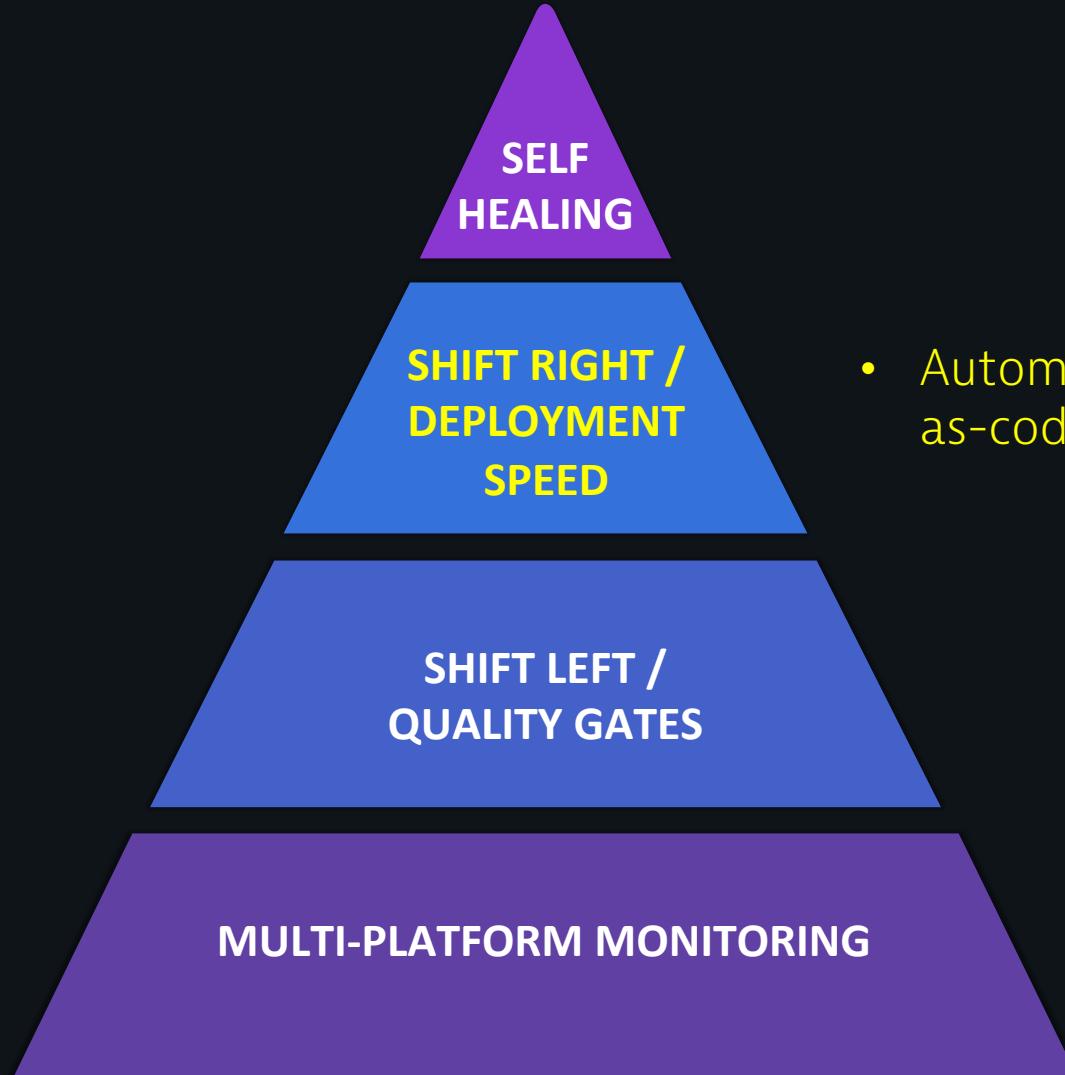
Metrics

- Response Time (Percentiles)
- Throughput & Per instance / Canary

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

order: Multiple service problems
Problem 886 detected at 15:38 (open for 7 minutes).

	Affected applications	-
	Affected services	1
	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 Impacted users		270k Affected service calls
--	---------------------	--	--------------------------------

[▼ Show more](#)

#2

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

#3

777a0ed445d3f5f2b

2	...
1	...
1	...
1	...
1	...
1	...

ay, 11:21

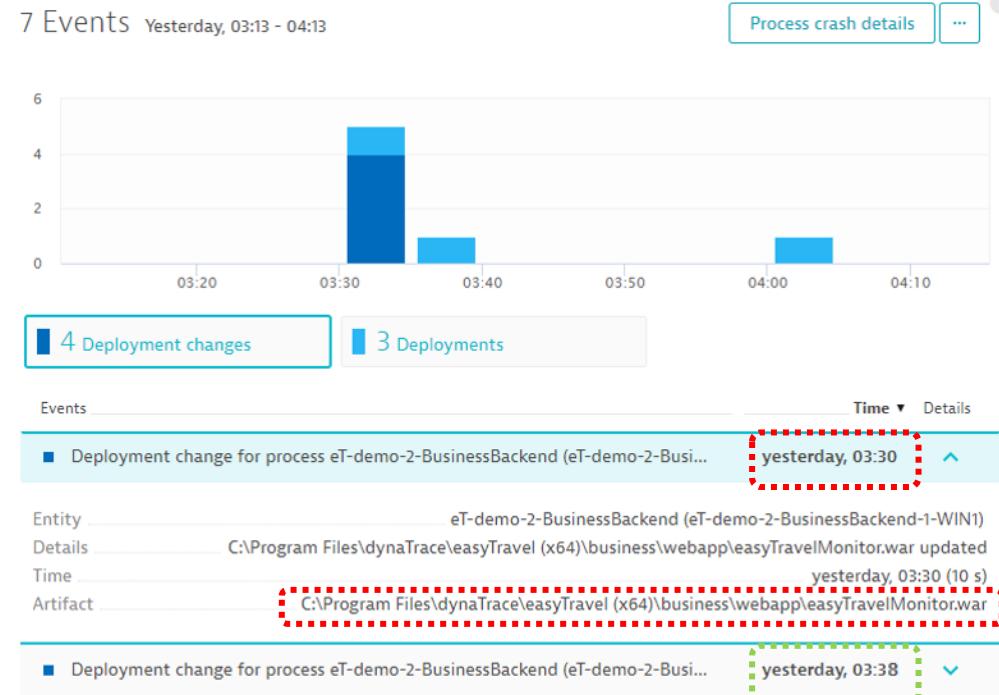
front-end-*
today, 11:21
Jenkins

777a0ed445d3f5f2b

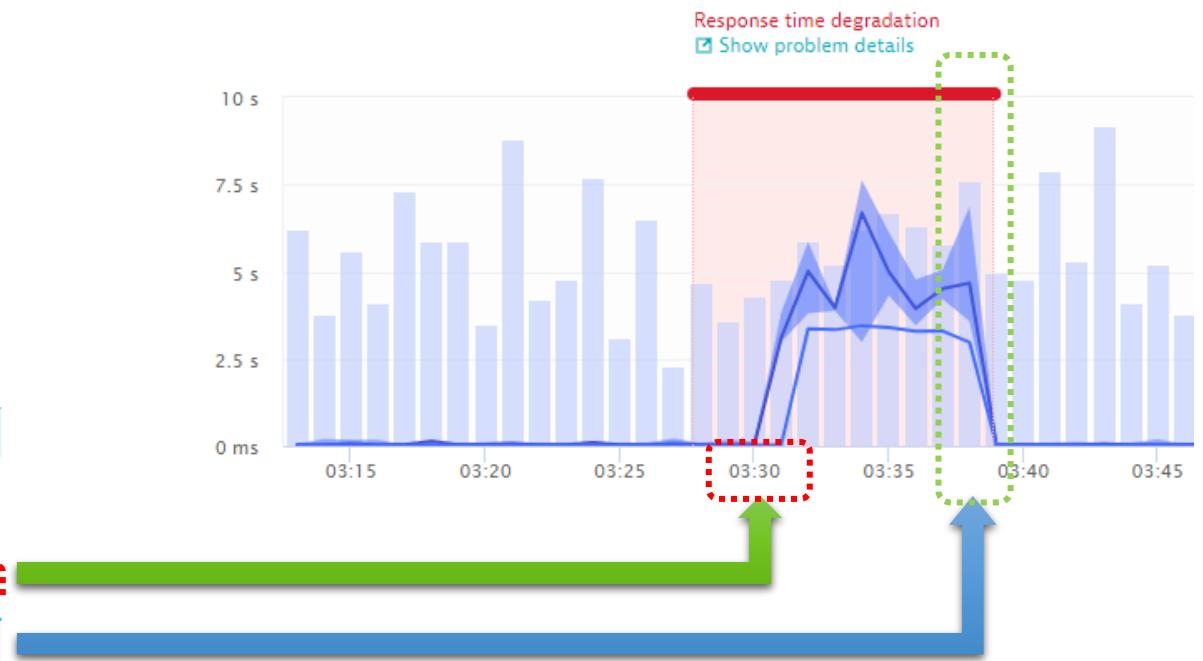
2	...
1	...
1	...
1	...
1	...



Provides more context to your environment



Response time



Key blue print – unbreakable continuous delivery model



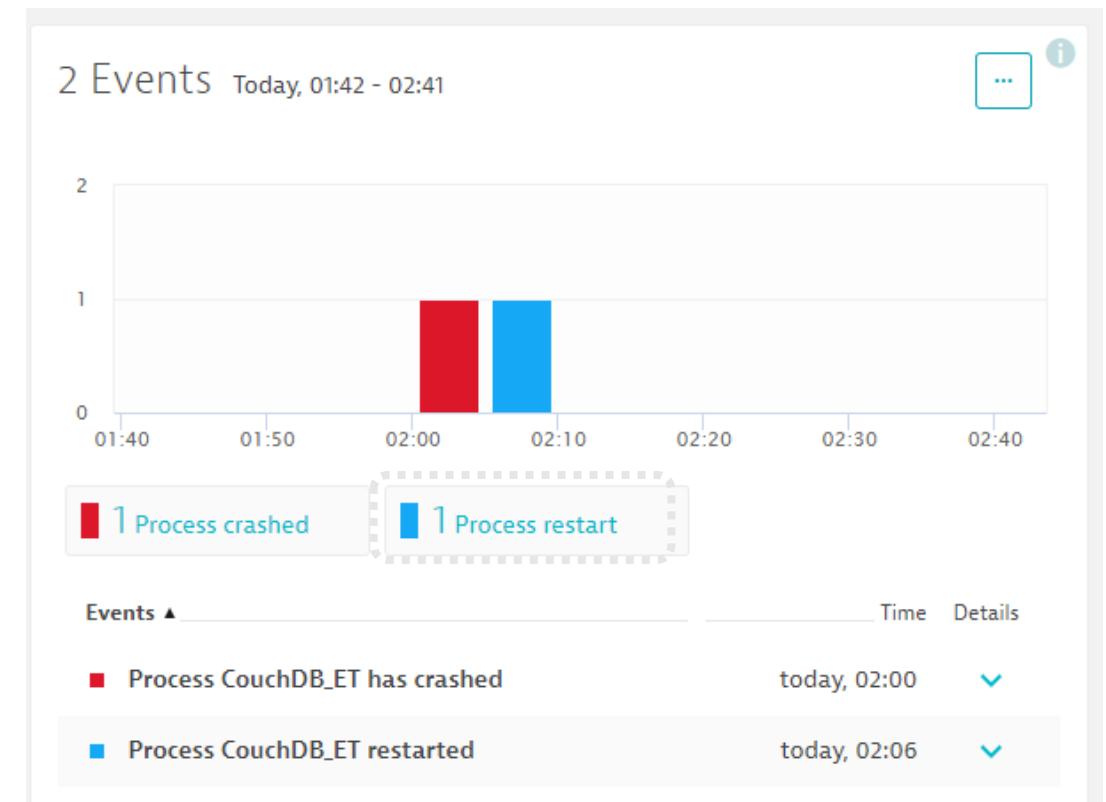
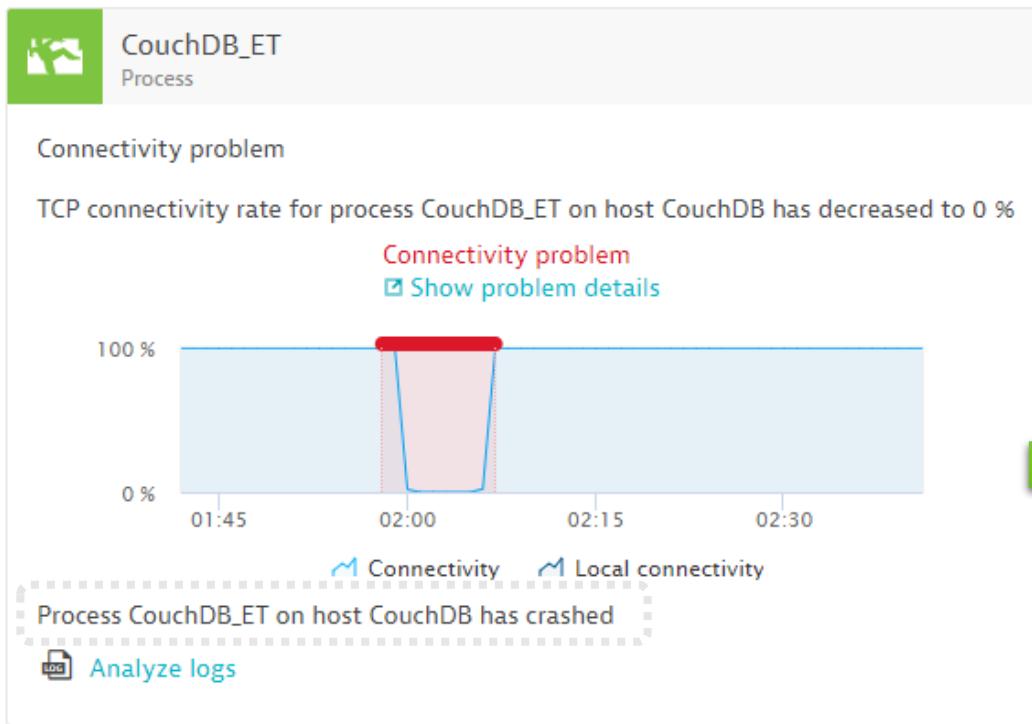
- Automate operations (self-healing) – auto-mitigate bad deployments in production



Use Case: Crash -> Restart

Root cause

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

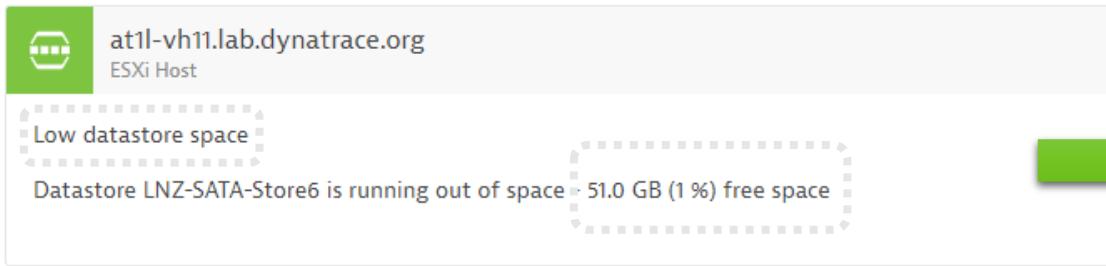




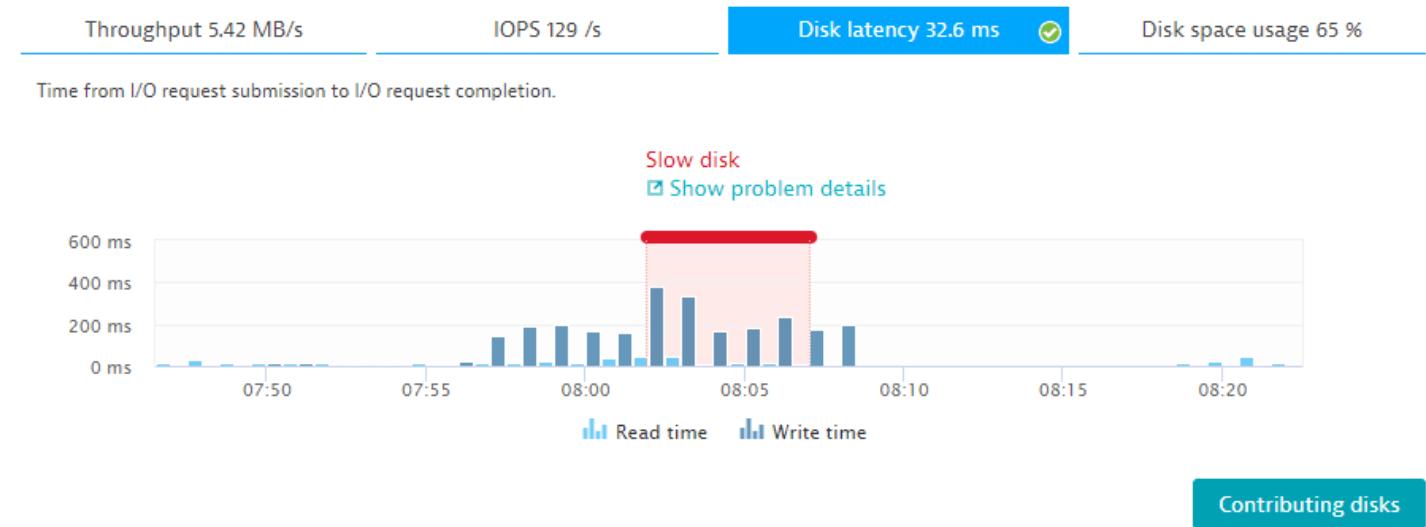
Use Case: Full or slow Disk -> Clean Up

at11-vh11.lab.dynatrace.org
ESXi Host

Low datastore space
Datastore LNZ-SATA-Store6 is running out of space 51.0 GB (1 %) free space

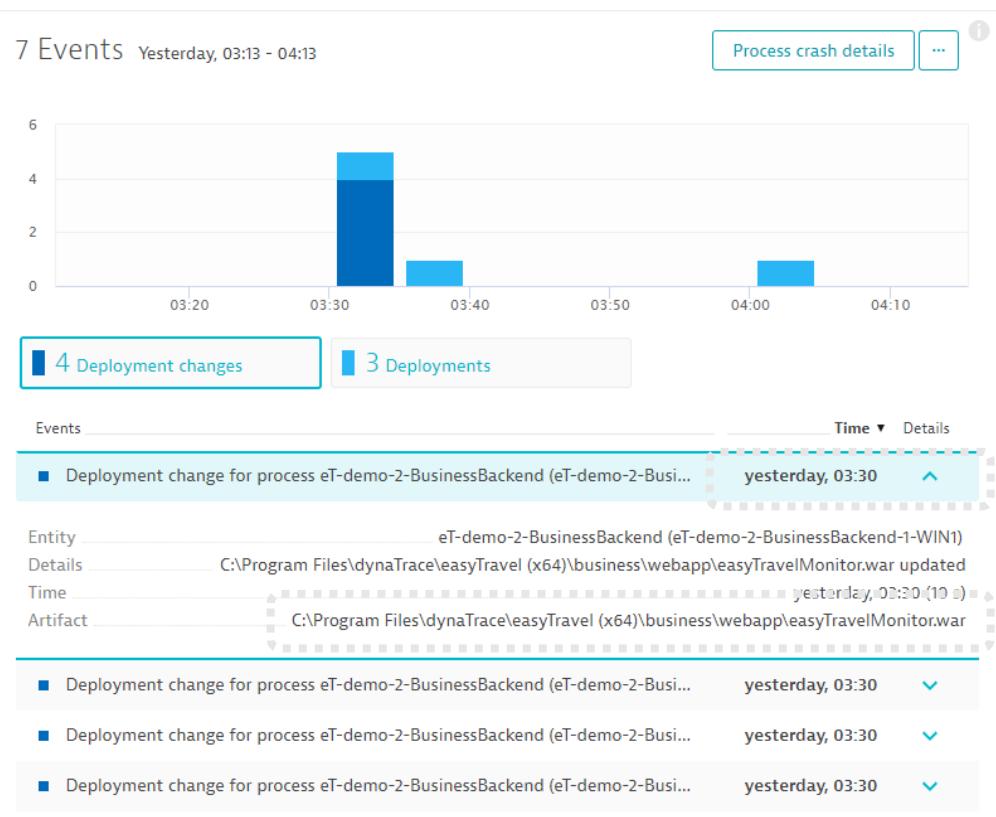


```
$ find ./my_dir -mtime +10 -type f -delete
```

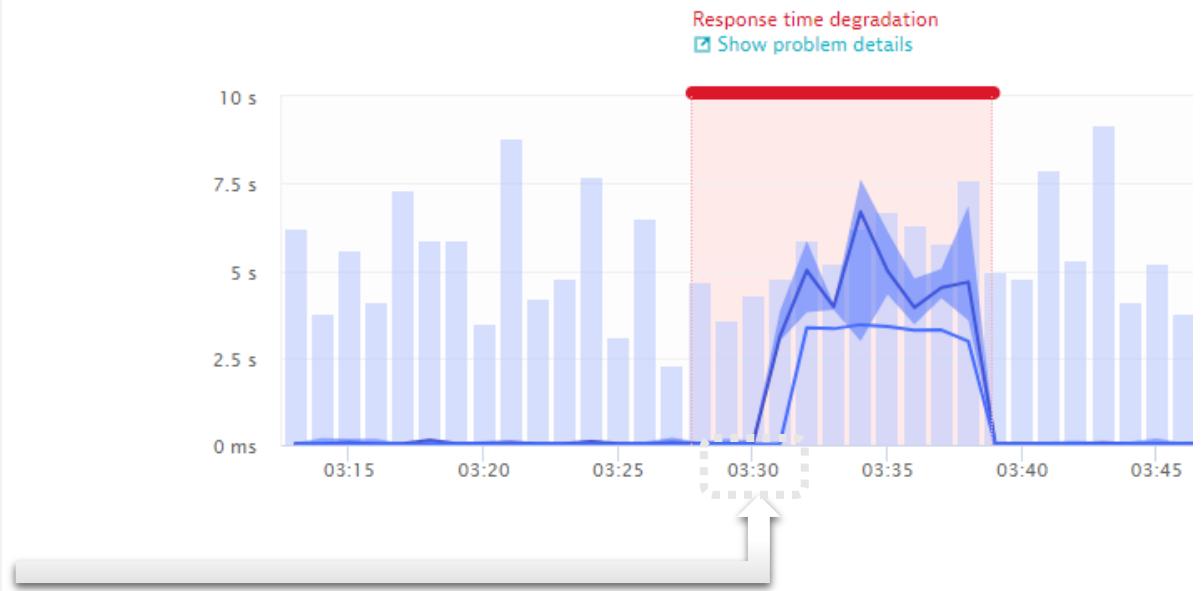




Use Case: Bad Configuration Changes -> Revert



Response time





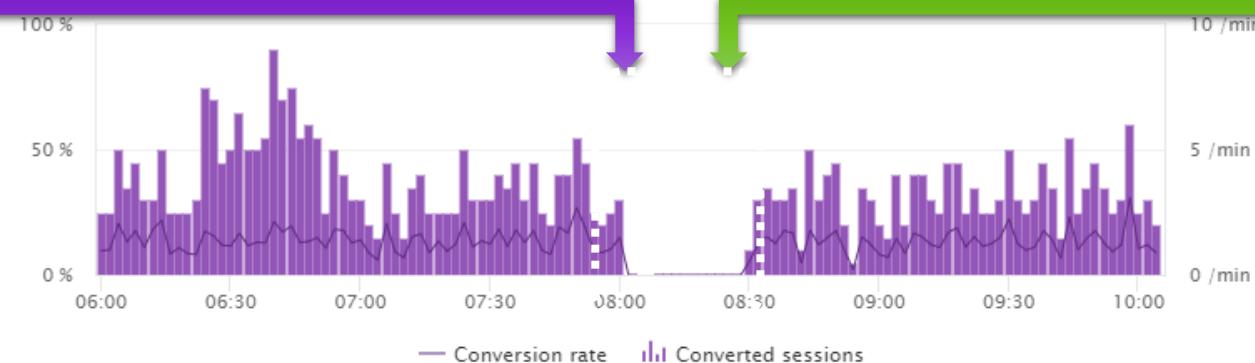
Use Case: End User Impact -> Reverse Blue / Green

Deploy Blue

Back to Green

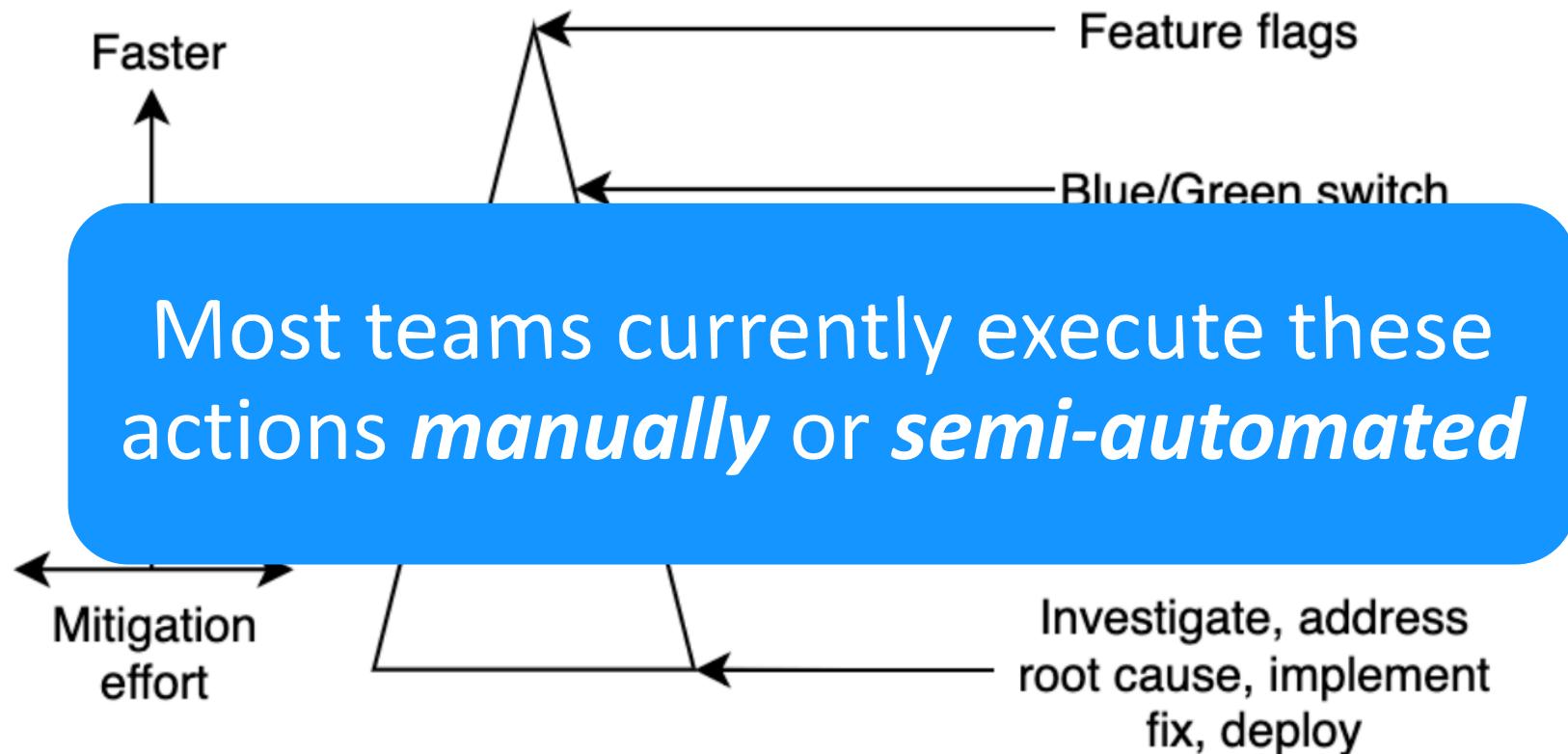
Conversion trend

Shows overall rate of success toward your conversion goals. Each converted session met at least one of your conversion goals.





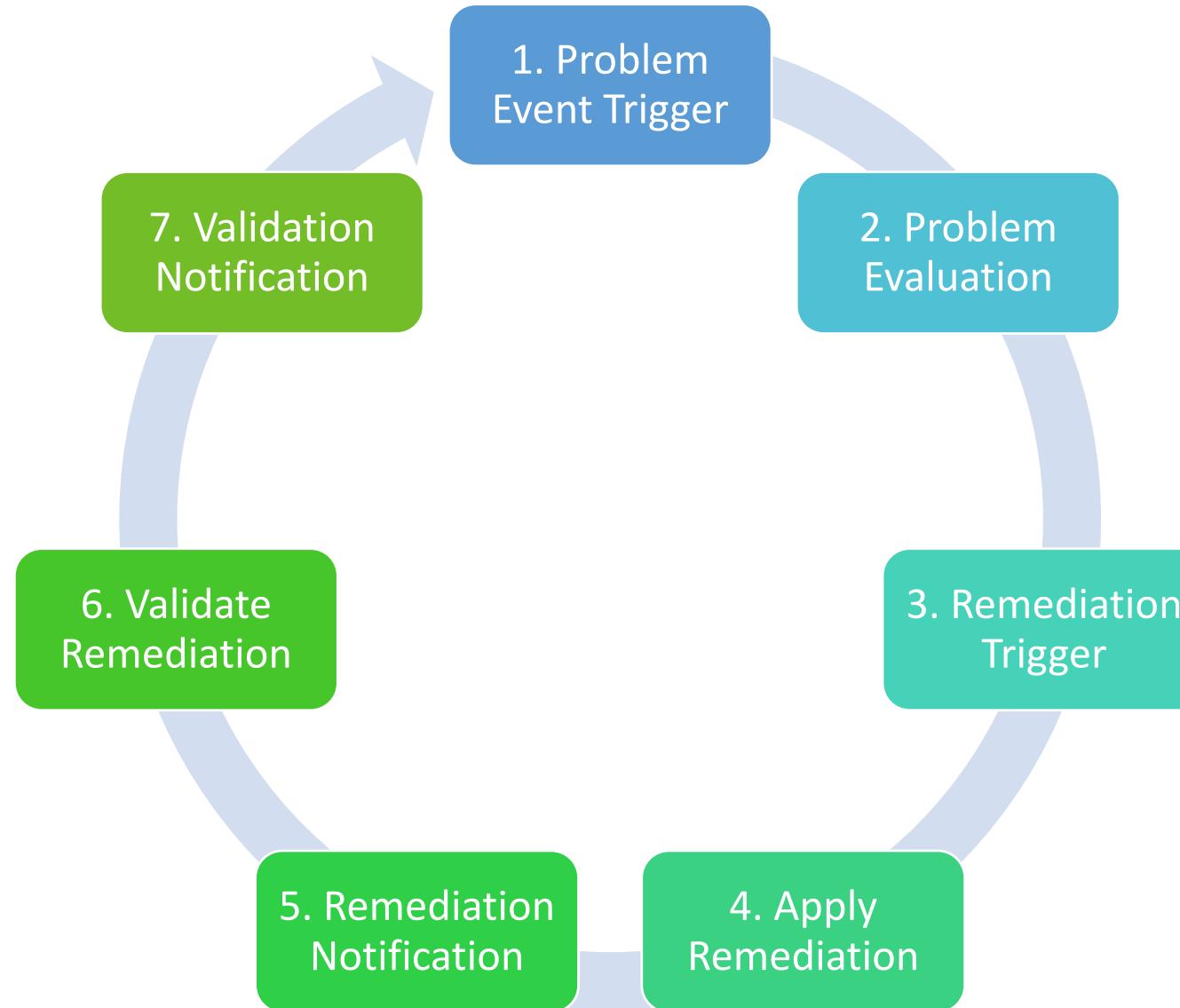
Time to take action



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



Auto Remediation Closed Loop System



1 User Impact
Problem Evolution



Application / Cloud
Infrastructure Monitoring Tool

```
"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!

2 Self-healing
Automation

2:00 a.m. escalate?



Auto-mitigate!



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



```
Getting Cluster Credentials
-----
[✓] saved kubeconfig as "/home/ubuntu/.kube/config"
=====
Finished provisioning AWS EKS Cluster
=====
Script start time : Fri Jun  7 20:32:57 UTC 2019
Script end time   : Fri Jun  7 20:48:25 UTC 2019
=====
Validating Kubectl configured to cluster
=====
Able to Connect to Cluster using kubectl.
```



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

```
=====
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/master
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/master
new remote: jyarbrough-workshop
Done.
Cloning https://github.com/dt-kube-demo/front-end
```



```
=====
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
```

The screenshot shows the GitHub organization 'jyarbrough-workshop' with the following details:

- deploy**: Forked from dt-kube-demo/deploy. Language: Apache-2.0. Stars: 34. Updated on Mar 26.
- customer-service**: Forked from dt-kube-demo/customer-service. Language: Java. Stars: 32. Updated on Mar 25.
- front-end**: Forked from dt-kube-demo/front-end. Language: HTML. Stars: 34. Updated on Mar 22.
- order-service**: Forked from dt-kube-demo/order-service.



Verify cluster is up

The diagram illustrates the verification process for an AWS cluster setup across four main services:

- VPC:** The first dashboard shows the VPC Dashboard with a search bar for "jahn" and a list of VPCs, including "jahn-demo-app" with ID "vpc-072c84f7e". A large blue arrow points to this dashboard.
- IAM Role:** The second dashboard shows the IAM Roles page with a search bar for "jahn" and a list of roles, including "jahn-demo-app-cluster" and "jahn-demo-app-workers". A large blue arrow points to this dashboard.
- EKS:** The third dashboard shows the Amazon Container Services EKS Clusters page for "jahn-demo-app-cluster". It displays general configuration details like Kubernetes Version 1.11, Platform Version eks.2, and an active status. A large blue arrow points to this dashboard.
- EC2:** The fourth dashboard shows the EC2 Instances page with a search bar for "jahn" and a list of instances, including "jahn-demo-app-bastion", "jahn-demo-app-workers", and two other entries. A large blue arrow points to this dashboard.



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.



```
=====  
Creating K8s namespaces ...  
namespace/cicd created  
namespace/staging created  
namespace/production created  
namespace/dynatrace created  
-----  
Setting up Jenkins ...  
-----  
Deploying Jenkins ...  
=====
```

```
=====  
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
```

```
=====  
About to setup demo app infrastructure with these parameters:  
"jenkinsPort": "24711",  
"jenkinsUser": "admin",  
"jenkinsPassword": "AiTx4u8VyUV8tCKk",  
"dynatraceTenant": "ibg73613",  
"dynatraceApiToken": "m3sTRYBnRSeaSbueCqh0F",  
"dynatracePaaSToken": "lojneX1rTx0kEJS1uJxEo",  
"dynatraceHostName": "ibg73613.live.dynatrace.com",  
"githubUserName": "pcjeffmac",  
"githubPersonalAccessToken": "0c04a430bd5b36c0d9",  
"githubUserEmail": "pcjeffmac@gmail.com",  
"githubOrg": "jyarbrough-workshop",  
Press ctrl-c to abort. Press any key to continue...
```

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



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



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



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



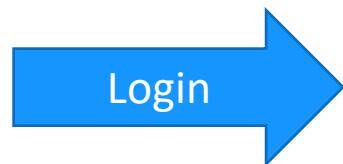
Log into Jenkins

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





Validate Jenkins

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

The screenshot shows the Jenkins management interface. On the left, the 'Manage Jenkins' menu is open, with 'Manage Plugins' selected. The main panel displays the 'Available' tab of the plugin manager. A single plugin, 'HTTP Request', is listed. It has a brief description: 'This plugin sends a http request to an url with some parameters'. Below the list are two buttons: 'Install without restart' and 'Download now and install after restart'. There is also a link to 'Update information'.



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

The screenshot shows the Jenkins web interface for managing credentials. On the left, there's a sidebar with various links like New Item, People, Build History, Manage Jenkins, My Views, Lockable Resources, Credentials (which is currently selected), System, Kubernetes, and New View. The main area is titled 'Credentials' and displays a table of existing credentials:

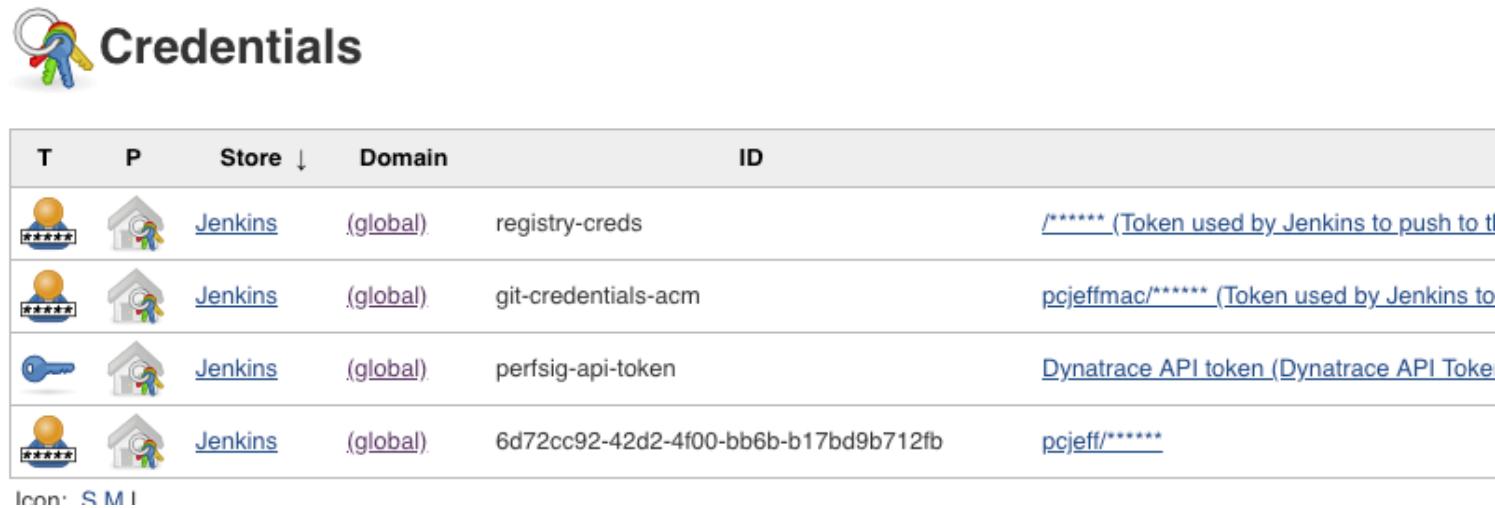
T	P	Store ↓	Domain	ID	Name
SSH	File	Jenkins	(global)	registry-creds	/***** (Token used by Jenkins to push to the container registry)
SSH	File	Jenkins	(global)	git-credentials-acm	pcjeffmac/***** (Token used by Jenkins to access the GitHub repositories)
Key	File	Jenkins	(global)	perfsig-api-token	Dynatrace API token (Dynatrace API Token used by the Performance Signature plugin)

Below the table, it says 'Stores scoped to Jenkins'. The bottom half of the screen shows the 'Add Credential' form for a 'Username with password' type. The form fields are: Kind (Username with password), Scope (Global (Jenkins, nodes, items, all child items, etc)), Username, Password, ID, and Description.



Copy credential hash

1. Select credentials and copy the ID.



The screenshot shows the Jenkins Credentials Manager page. At the top, there's a search bar with a magnifying glass icon and a 'Credentials' button. Below the header is a table with the following columns: Type (T), Provider (P), Store, Domain, and ID. The table contains four rows of data:

T	P	Store ↓	Domain	ID
Token	Jenkins	(global)	registry-creds	/***** (Token used by Jenkins to push to the Docker registry)
Token	Jenkins	(global)	git-credentials-acm	pcjeffmac/***** (Token used by Jenkins to access GitHub)
Key	Jenkins	(global)	perfsig-api-token	Dynatrace API token (Dynatrace API Token)
Token	Jenkins	(global)	6d72cc92-42d2-4f00-bb6b-b17bd9b712fb	pcjeff/*****

Icon: S M I

Deploy the application



Lets run the “deploy-staging” job

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

The screenshot shows the Jenkins dashboard with the following elements:

- Left Sidebar:** Includes links for New Item, People, Build History, Manage Jenkins, My Views, Lockable Resources, Credentials, and New View.
- Build Queue:** Displays "No builds in the queue."
- Build Executor Status:** Displays "No builds in the queue."
- Middle Panel:** Shows a list of pipelines:
 - S: andig_test (Status: Green)
 - W: deploy-production (Status: Green)
 - S: deploy-service (Status: Red)
 - W: deploy-staging (Status: Yellow)
 - S: (Status: Grey)
- Context Menu for deploy-staging:** A dropdown menu appears over the deploy-staging pipeline entry, containing:
 - Changes
 - Build with Parameters** (highlighted with a blue arrow)
 - Delete Pipeline
 - Configure
 - Full Stage View
- #1** and **#2**: Blue arrows indicating the steps: #1 points from the sidebar to the pipeline list; #2 points from the pipeline list to the context menu.

Pipeline deploy-staging

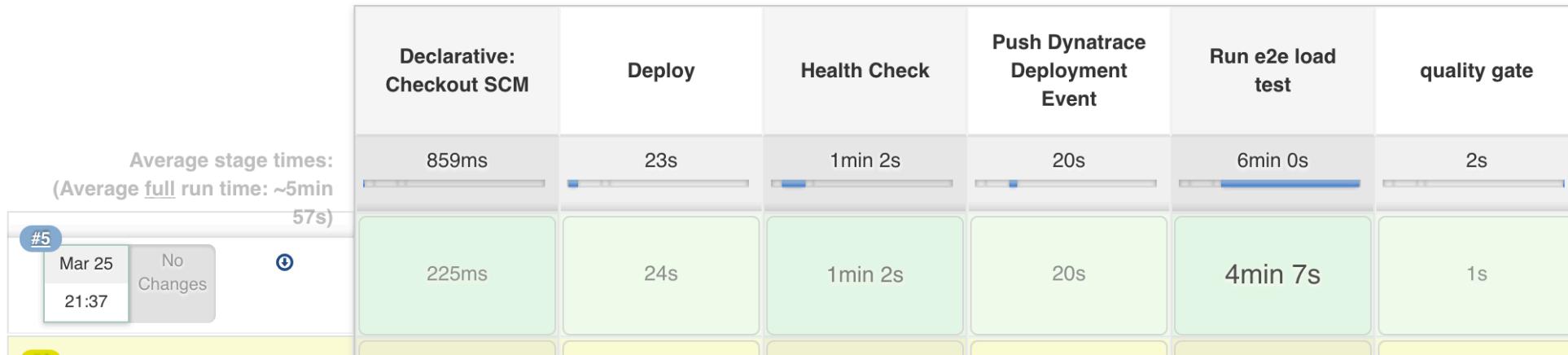
This build requires parameters:

frontendversion	<input type="text" value="1"/>	image tag when custom version option chosen
orderserviceversion	<input type="text" value="1"/>	image tag when custom version option chosen
customerserviceversion	<input type="text" value="1"/>	image tag when custom version option chosen
catalogserviceversion	<input type="text" value="1"/>	image tag when custom version option chosen
registry	<input type="text" value="robjahn"/>	Dockerhub account with Docker Images



Lets run the “deploy-staging” job

- View status and console log



Jenkins

Jenkins ➔ deploy-staging ➔ #5

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Delete Build

Console Output

```
Started by user admin
Obtained Jenkinsfile.DeployStaging from git 1
Running in Durability level: MAX_SURVIVABILITY
Loading library dynatrace@master
Attempting to resolve master from remote refe
> git --version # timeout=10
> git ls-remote -h https://github.com/dynatr
Found match: refs/heads/master revision 3f494
```



Lets run the “deploy-production” job

- Choose “build with parameters”

Jenkins

New Item

People

Build History

Manage Jenkins

My Views

Lockable Resources

Credentials

New View

All

#1

#2

andig_test

Changes

Build with Parameters

Delete Pipeline

Pipeline deploy-production

This build requires parameters:

frontendversion	1	image tag when custom version option chosen
orderserviceversion	1	image tag when custom version option chosen
customerserviceversion	1	image tag when custom version option chosen
catalogserviceversion	1	image tag when custom version option chosen
registry	robjahn	Dockerhub account with Docker Images

Build



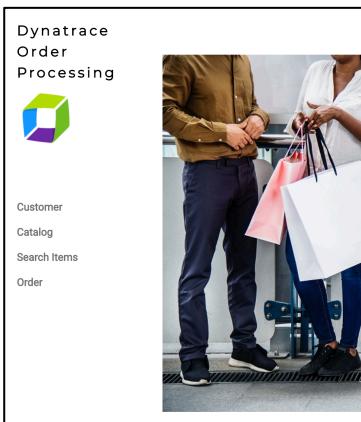
Log into bastion

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

Rule

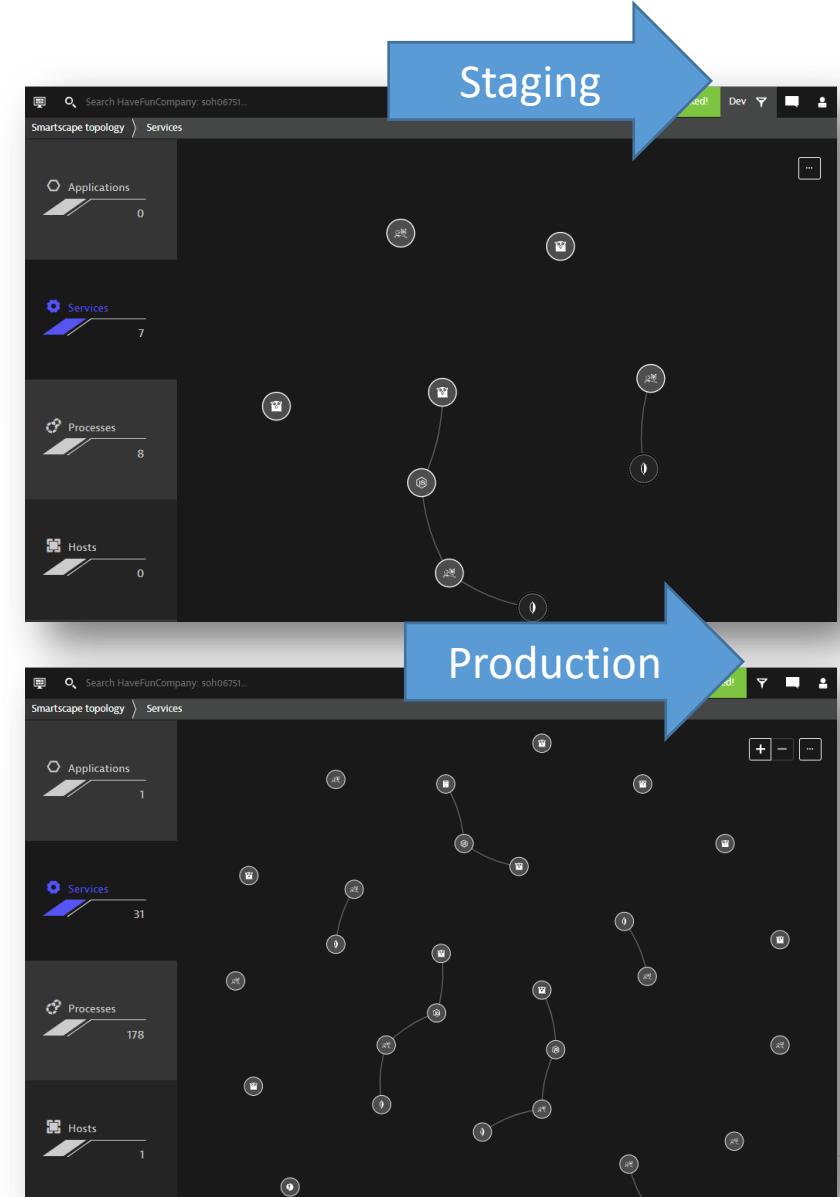
Services tagged with 'dt-kube-demo-environment:production'

Rule applies to Services of the following process group, service type, technology, and topology

Conditions

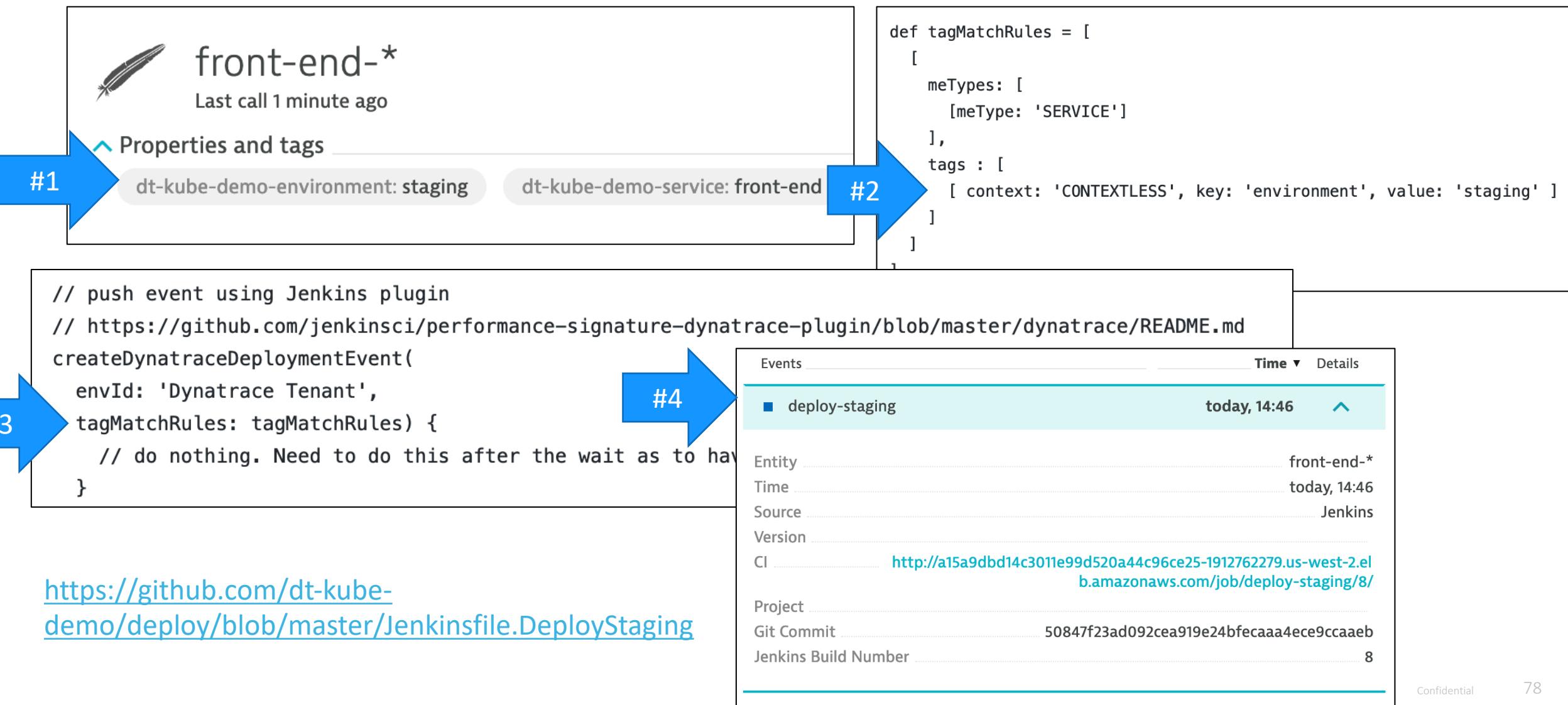
Service tags equals dt-kube-demo-environment production

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





In Dynatrace Deployment Event shows up on Deployed Service





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



```
        }
    }"""
//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/"
}
```



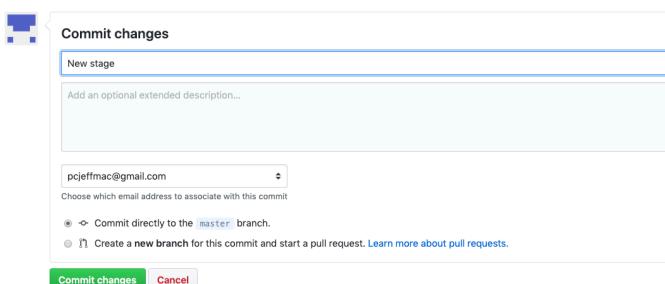
Add A Deployment event (http request)

- 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

168 lines (151 sloc) | 6.11 KB

Raw Blame History

```
1 @Library('dynatrace@master') _
2
3 def tagMatchRules = [
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84 stage('Push Dynatrace Deployment Event') {
85   steps {
86     script {
87       echo "Waiting for the service to be tagged..."
88       sleep 20
89
90       // push event using Jenkins plugin
91       // https://github.com/jenkinsci/performance-signature-dynatrace-plugin/blob/master/dynatrace/README.md
92       createDynatraceDeploymentEvent(
93         envId: 'Dynatrace Tenant',
94         tagMatchRules: tagMatchRules,
95         customProperties: [key: 'frontendversion', value: "${params.frontendversion}"],[key: 'orderversion', value: "${params.orderversion}"]
96       )
97       // do nothing. Need to do this after the wait as to have service up
98     }
99   }
100 }
101
102
103 stage('Event-Post-Service-Deployment') { //Dynatrace POST action for deployment Event - requires HTTP Request Plugin
104   steps {
105     script {
106       def body = """{"eventType": "CUSTOM_DEPLOYMENT",
107         "attachRules": {
108           "tagRule": {
109             "meType": "SERVICE",
110             "tags": [
111               {
112                 "context": "CONTEXTLESS",
113                 "key": "dt-kube-demo-service",
114                 "value": "front-end"
115               }
116             ]
117           }
118         }
119       """
120     }
121   }
122 }
```





Lets run the “deploy-staging” job

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

The screenshot shows the Jenkins interface for the "Pipeline deploy-staging" job. On the left, the Jenkins sidebar includes links for New Item, People, Build History, Manage Jenkins, My Views, Lockable Resources, Credentials, and New View. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status.

The main area displays a list of stages:

S	W	Name ↓
●	☀️	andig_test
●	☀️	deploy-production
●	☁️	deploy-service
●	☀️	deploy-staging
●	🌧️	

A blue arrow labeled "#1" points from the sidebar to the "deploy-staging" stage. A second blue arrow labeled "#2" points from the stage list to the context menu for the "deploy-staging" stage. The context menu is open, showing options: Changes, Build with Parameters (which is highlighted), Delete Pipeline, Configure, and Full Stage View. To the right of the menu is a "Build" button.

On the right, the "Pipeline deploy-staging" configuration page is shown. It lists required parameters:

- frontendversion: 1 (image tag when custom version chosen)
- orderserviceversion: 1 (image tag when custom version chosen)
- customerserviceversion: 1 (image tag when custom version chosen)
- catalogserviceversion: 1 (image tag when custom version chosen)
- registry: robjahn (Dockerhub account with Docker Images)



In Dynatrace Deployment Event shows up on Deployed Service

front-end-*

Last call 1 minute ago

Properties and tags

#1

dt-kube-demo-environment: staging dt-kube-demo-service: front-end

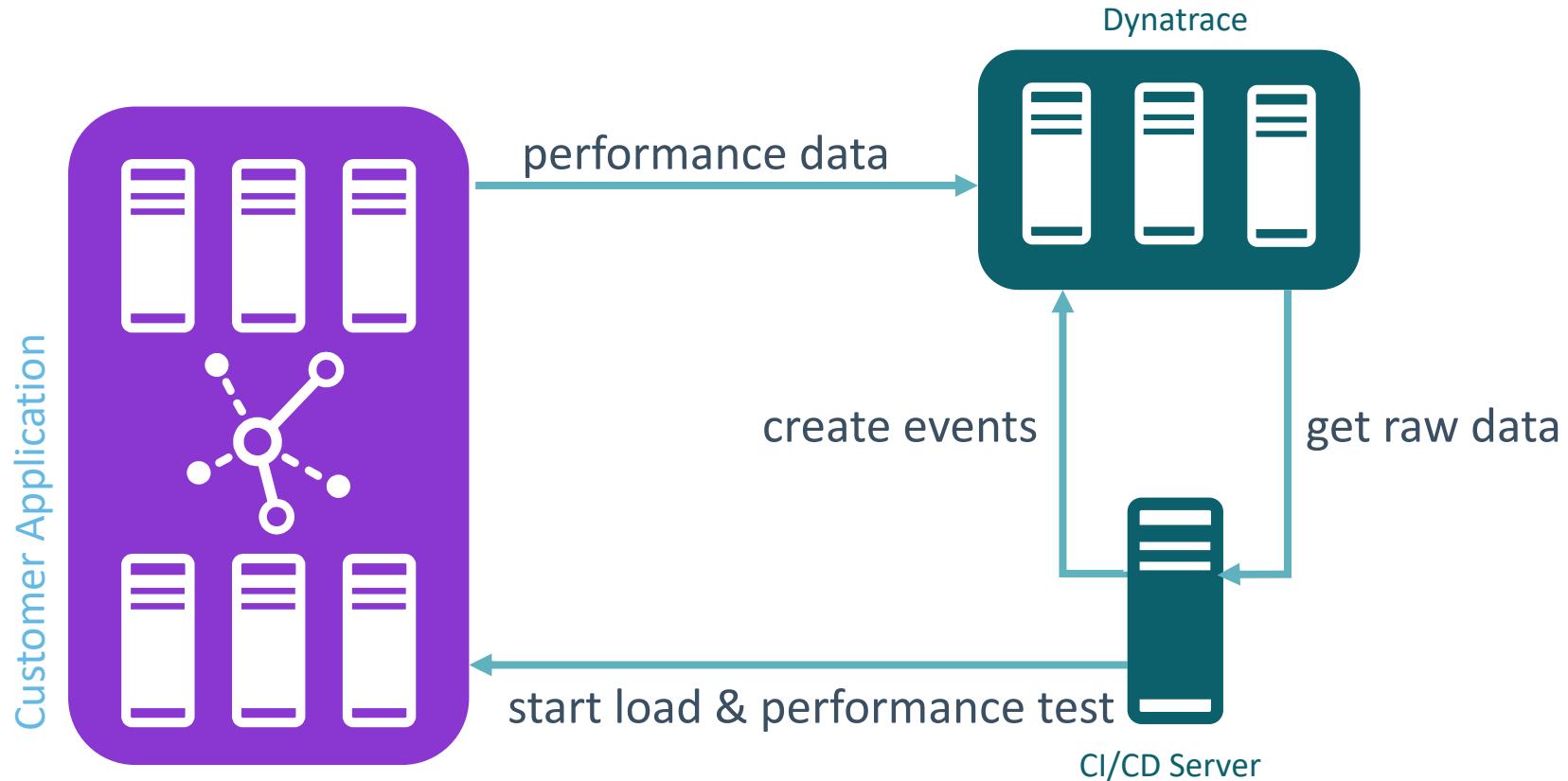
```
102  
103 stage('Event-Post-Service-Deployment') { //Dynatrace POST action for deployment Event - requires HTTP Request Plugin  
104     steps {  
105         script {  
106             def body = """{"eventType": "CUSTOM_DEPLOYMENT",  
107                 "attachRules": {  
108                     "tagRule" : {  
109                         "meTypes" : "SERVICE",  
110                         "tags" : [  
111                             {  
112                             "context" : "CONTEXTLESS",  
113                             "key" : "dt-kube-demo-service",  
114                             "value" : "front-end"  
115                         }  
116                     ]  
117                 }  
118             },  
119         }  
120     }  
121 }
```

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

Events	Time ▾	Details
■ deploy-staging - 2 Staging (http)	today, 17:57	▲
Entity	front-end-*	
Time	today, 17:57	
Source	Jenkins	
Version	2	
CI	http://ac2d1cdb8896611e9851d165c0af619d-520118970.us-east-1.elb.amazonaws.com:24711/job/deploy-staging/2/	
Remediation	https://ansible.pcjeffint.com/#/templates/job_template/7	
Project	pipeline	
Build URL	http://ac2d1cdb8896611e9851d165c0af619d-520118970.us-east-1.elb.amazonaws.com:24711/job/deploy-staging/2/	
Environment	Staging	
Jenkins Build Number	2	
Job URL	http://ac2d1cdb8896611e9851d165c0af619d-520118970.us-east-1.elb.amazonaws.com:24711/job/deploy-staging/	

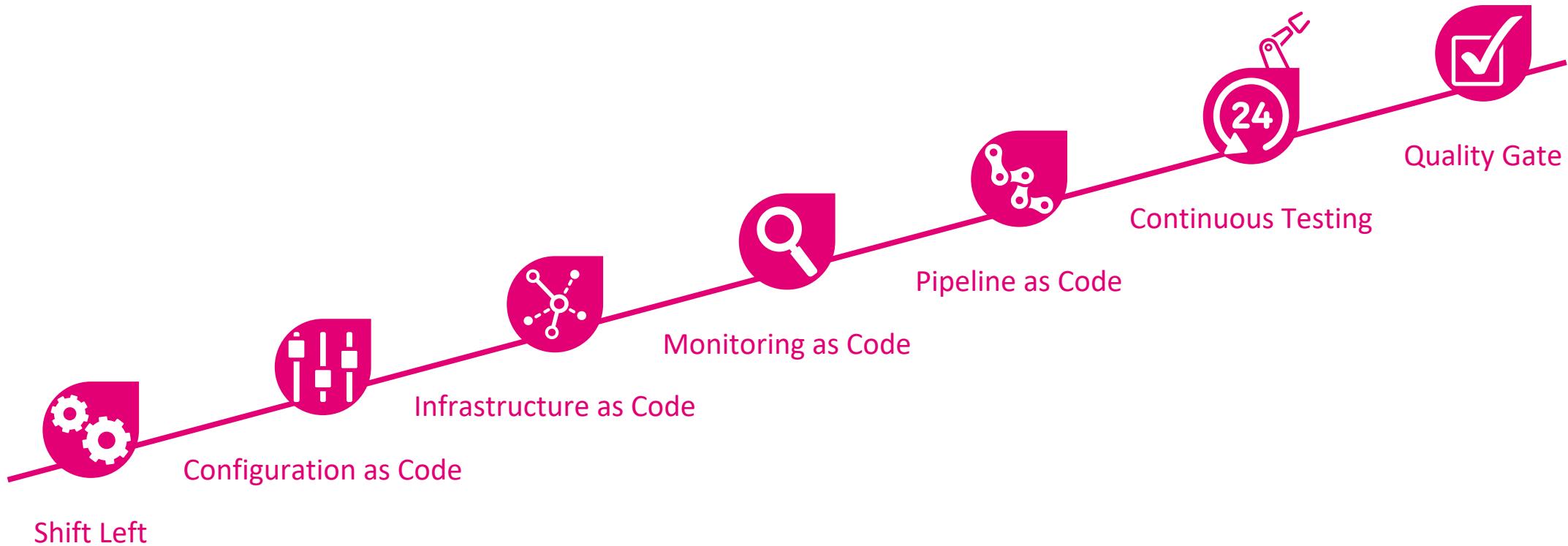


Performance Signature: Architecture





Performance Signature: Concepts

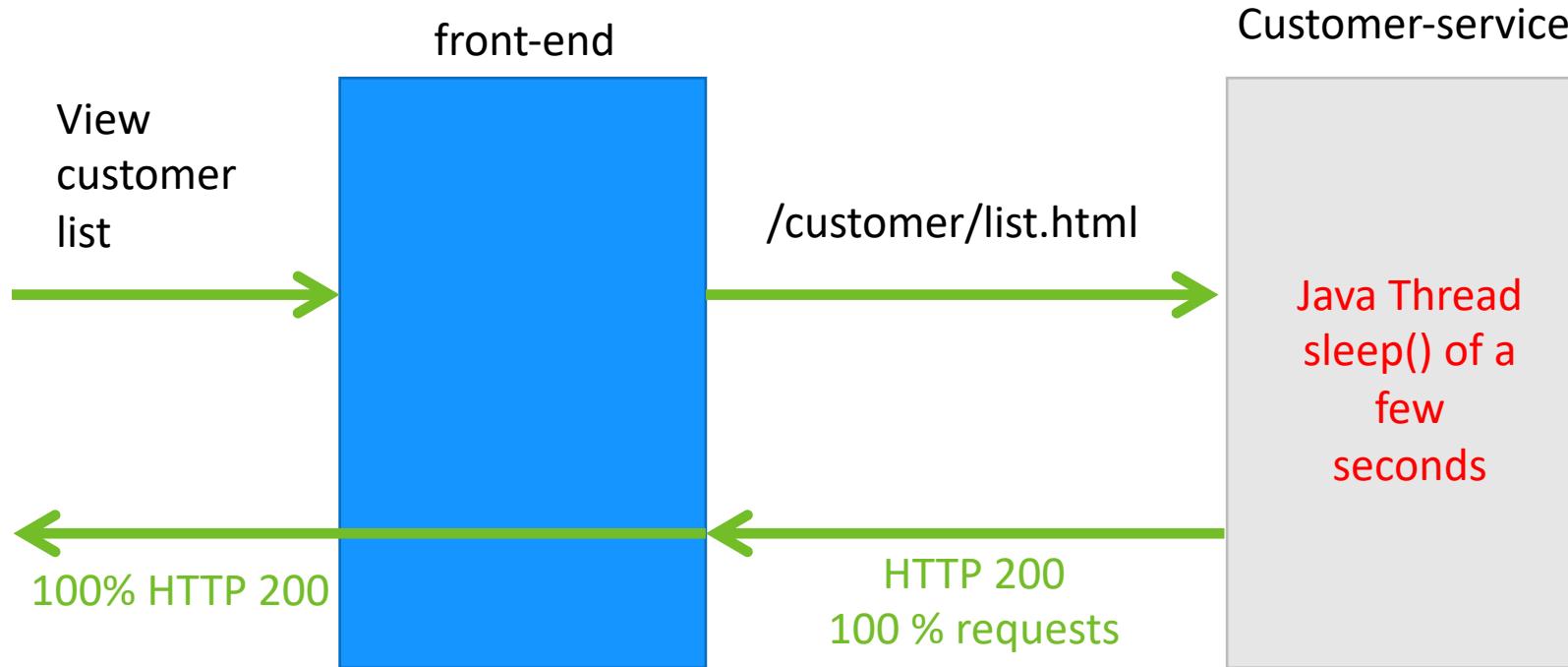


Lets prevent an issue

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

The screenshot illustrates the Jenkins Pipeline interface for the "deploy-staging" job. It shows two main steps: #1 selecting the job and #2 triggering it with parameters.

Step #1: The user has selected the "deploy-staging" pipeline from the list of available jobs. A blue arrow labeled "#1" points to the job name "deploy-staging".

Step #2: A context menu is open over the "deploy-staging" job entry, with the "Build with Parameters" option highlighted. A blue arrow labeled "#2" points to this menu item. The menu also includes options for "Changes", "Delete Pipeline", "Configure", and "Full Stage View".

Pipeline deploy-staging Configuration:

Parameter	Value	Description
frontendversion	1	image tag when custom version chosen
orderserviceversion	1	image tag when custom version chosen
customerserviceversion	2	image tag when custom version chosen
catalogserviceversion	1	image tag when custom version chosen
registry	robjahn	Dockerhub account with Docker Images

Build Button: A large blue "Build" button is located at the bottom right of the configuration panel.

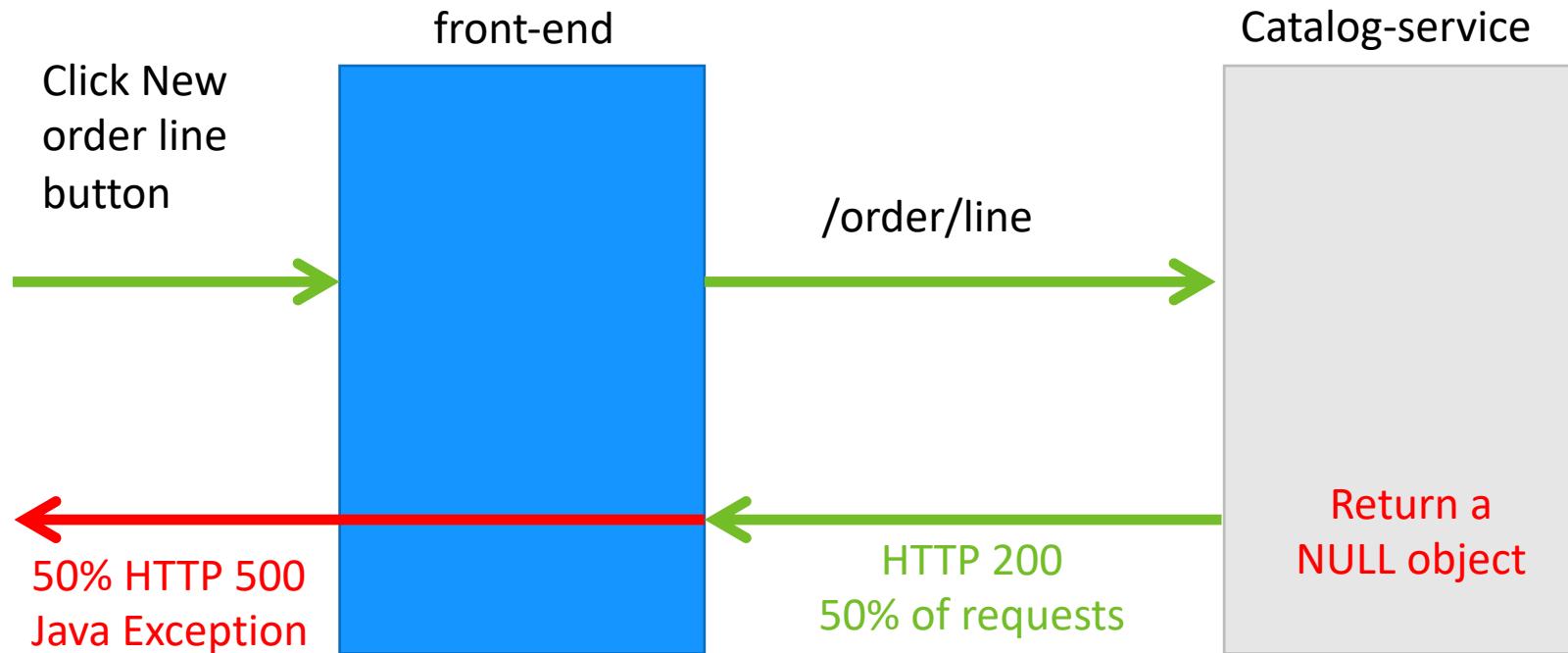


Review

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



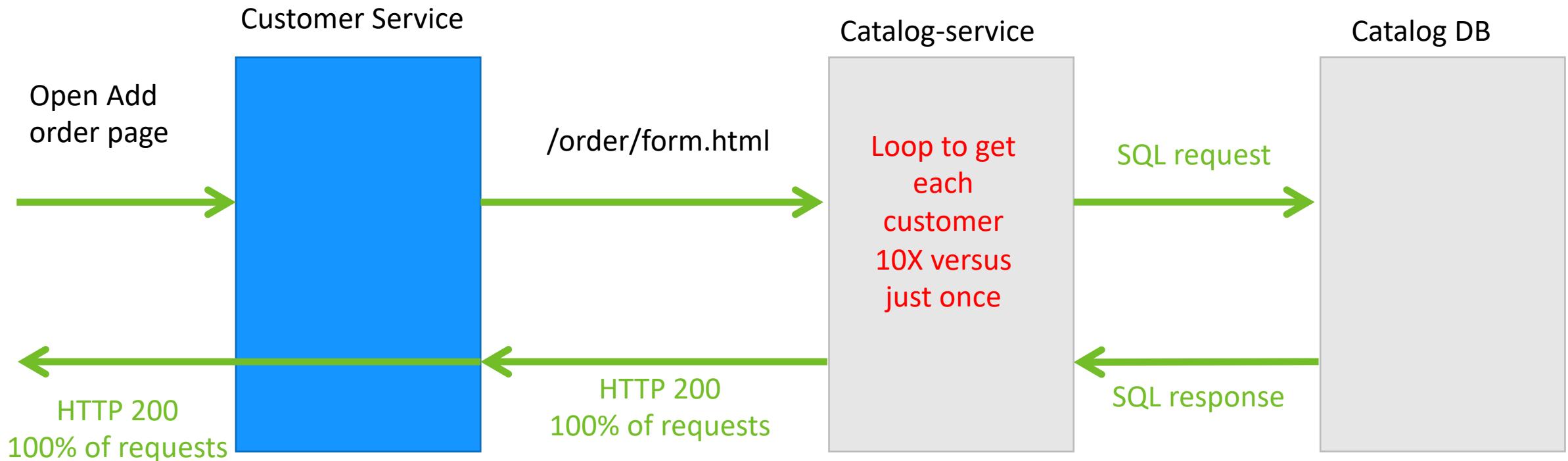
Scenario 2: Backend has exceptions in the code



- 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 response 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

The screenshot illustrates the Jenkins Pipeline interface for the "deploy-staging" pipeline. It shows the Jenkins dashboard on the left and the pipeline configuration on the right.

Jenkins Dashboard (Left):

- New Item
- People
- Build History
- Manage Jenkins
- My Views
- Lockable Resources
- Credentials
- New View

Pipeline Configuration (Right):

Pipeline deploy-staging

This build requires parameters:

frontendversion	1	image tag when custom version chosen
orderserviceversion	2	image tag when custom version chosen
customerserviceversion	1	image tag when custom version chosen
catalogserviceversion	1	image tag when custom version chosen
registry	robjahn	Dockerhub account with Docker Images

Actions:

- Changes
- Build with Parameters** (highlighted)
- Delete Pipeline
- Configure
- Full Stage View

Annotations:

- #1 points to the "Build with Parameters" option in the pipeline menu.
- #2 points to the "orderserviceversion" parameter value "2" in the configuration table.



Review

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

Automated Operations

Self-Healing



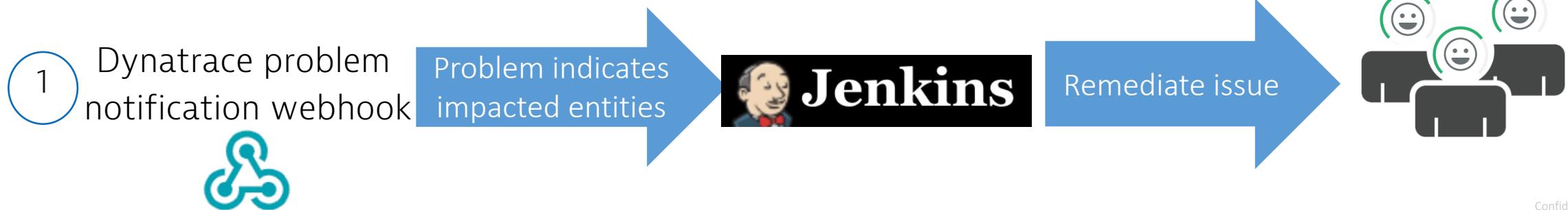
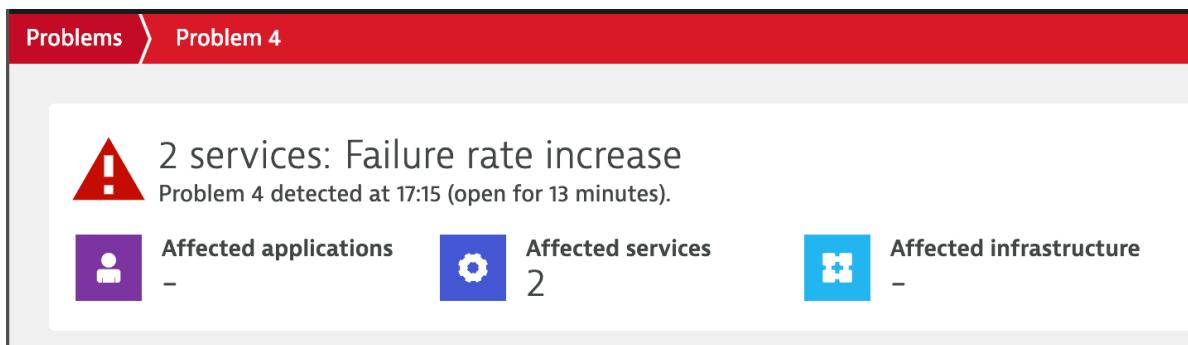
Self-healing Automation

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

Problems > Problem 4

! 2 services: Failure rate increase
Problem 4 detected at 17:15 (open for 13 minutes).

Affected applications - Affected services 2 Affected infrastructure -





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

The screenshot shows the Jenkins interface for a "Pipeline load-test" job. On the left, there's a sidebar with various links: Back to Dashboard, Status, Changes, Build with Parameters (which has a large blue arrow pointing to it), Delete Pipeline, Configure, Full Stage View, and Rename. The main content area is titled "Pipeline load-test". It says "This build requires parameters:" followed by "namespace production" with a dropdown arrow. Below that is a "Build" button. The URL in the browser bar is "Jenkins > load-test".

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

Severity rule

Availability alert (Immediate; dt-kube-demo-environment: production)

Problem severity level

Availability

Send a notification if a problem remains open longer than 0 minutes.

Filter problems by tag

Only include entities that have any tags

Select tag

dt-kube-demo-environment:production

Create tag filter

Remove filter

#2

dt-kube-demo

Create one or more filter rules for this alerting profile. Problems are filtered based on severity level, tagged impact, and problem duration.

Define severity rules for profile

Create alerting rule

Severity rule

Availability alert (Immediate; dt-kube-demo-environment: production)

Error alert (Immediate; dt-kube-demo-environment: production)

Slowdown alert (After 30 mins; dt-kube-demo-environment: production)

Resource alert (After 30 mins; dt-kube-demo-environment: production)

Custom alert (Immediate; dt-kube-demo-environment: production)



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

Settings > Anomaly detection > Services

Settings

- Monitoring
- Processes and containers
- Web & mobile monitoring
- Cloud and virtualization
- Server-side service monitoring
- Log Analytics
- Anomaly detection
- Applications
- Services**
- Database services

Anomaly detection for services

Dynatrace automatically detects service-related performance anomalies such as response time degradations and failure rate increases. Use these settings to configure detection sensitivity, set alert thresholds, or disable alerting for certain services.

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:

Reference period



Dynatrace problem notifications: Step 1 of 3

Get the URL to your 'deploy-production' job

The screenshot shows a web browser window with the following details:

- Address Bar:** Not Secure | a15a9dbd14c3011e99d520a44c96ce25-1912762279.us-west-2.elb.amazonaws.com/job/deploy-production/
- Tab Bar:** Apps, New Tab, Speechnotes | Sp..., safia-habib/Unbre..., Continuous Integr..., Salesforce - Unlim..., GTM Resource Ce..., dt-kube-demo
- Header:** Jenkins
- Breadcrumbs:** Jenkins > deploy-production
- Left Sidebar:** Back to Dashboard, Status, Changes, Build with Parameters, Delete Pipeline, Configure.
- Center Content:** Pipeline deploy-production, Recent Changes (link).

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

Integrate with other notification systems

Integrate Dynatrace problem notifications with your organization's existing incident-management system or team-collaboration channel. [Alerting profiles](#) are used within problem integrations to filter the total number of alerts to a subset that is relevant for your team.

 OpsGenie Integrate with an OpsGenie incident management platform.	 VictorOps Integrate with the VictorOps incident management platform.	 PagerDuty Integrate with the PagerDuty incident management platform.	 Slack Integrate with a Slack team collaboration channel.
 HipChat Integrate with a HipChat team collaboration chat.	 ServiceNow Integrate with a ServiceNow enterprise service management.	 Ansible Tower Integrate with the Ansible Tower deployment management platform.	 Email Notify other systems via email.
 JIRA Integrate with JIRA to automatically create issues.	 Trello Integrate with Trello to automatically create cards on your board.	 xMatters Integrate with the xMatters incident management platform.	 Custom integration Integrate by building your own web hook. #2



Dynatrace problem notifications: Step 3 of 3

Edit custom integration

Name
dt-kube-demo

Webhook URL
http://a9352ae514a7111e99d520a44c96ce25-47275794.us-west-2.elb.amazonaws.com/job/deploy-production/buildWithParameters?token=DT&cause=auto-remediation-from-Dynatrace

Accept any SSL certificate (Self signed or invalid)

Additional HTTP headers
+ Add header *0 Create basic authorization header

Custom payload
{
"State":"{State}",
"ProblemID":"{ProblemID}",
"ProblemTitle":"{ProblemTitle}"
}

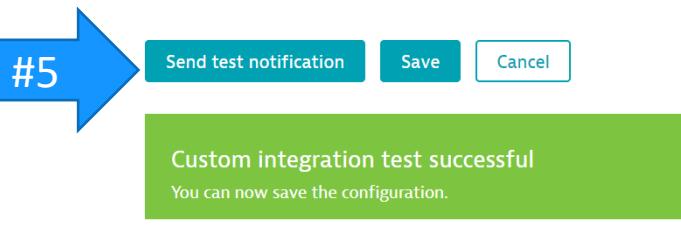
Alerting profile
dt-kube-demo

Select an alerting profile to control the delivery of notifications related to this integration.

#1 **#2** **#3** **#4** **#5**

Send test notification **Save** **Cancel**

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

Jenkins

New Item

People

Build History

Manage Jenkins

My Views

Lockable Resources

Credentials

New View

All

Name ↓

andig_test

deploy-production

#1

#2

Changes

Build with Parameters

Delete Pipeline

Pipeline deploy-production

This build requires parameters:

frontendversion	1	image tag when custom version chosen
orderserviceversion	2	image tag when custom version chosen
customerserviceversion	1	image tag when custom version chosen
catalogserviceversion	1	image tag when custom version chosen
registry	robjahn	Dockerhub account with Docker Images

Build

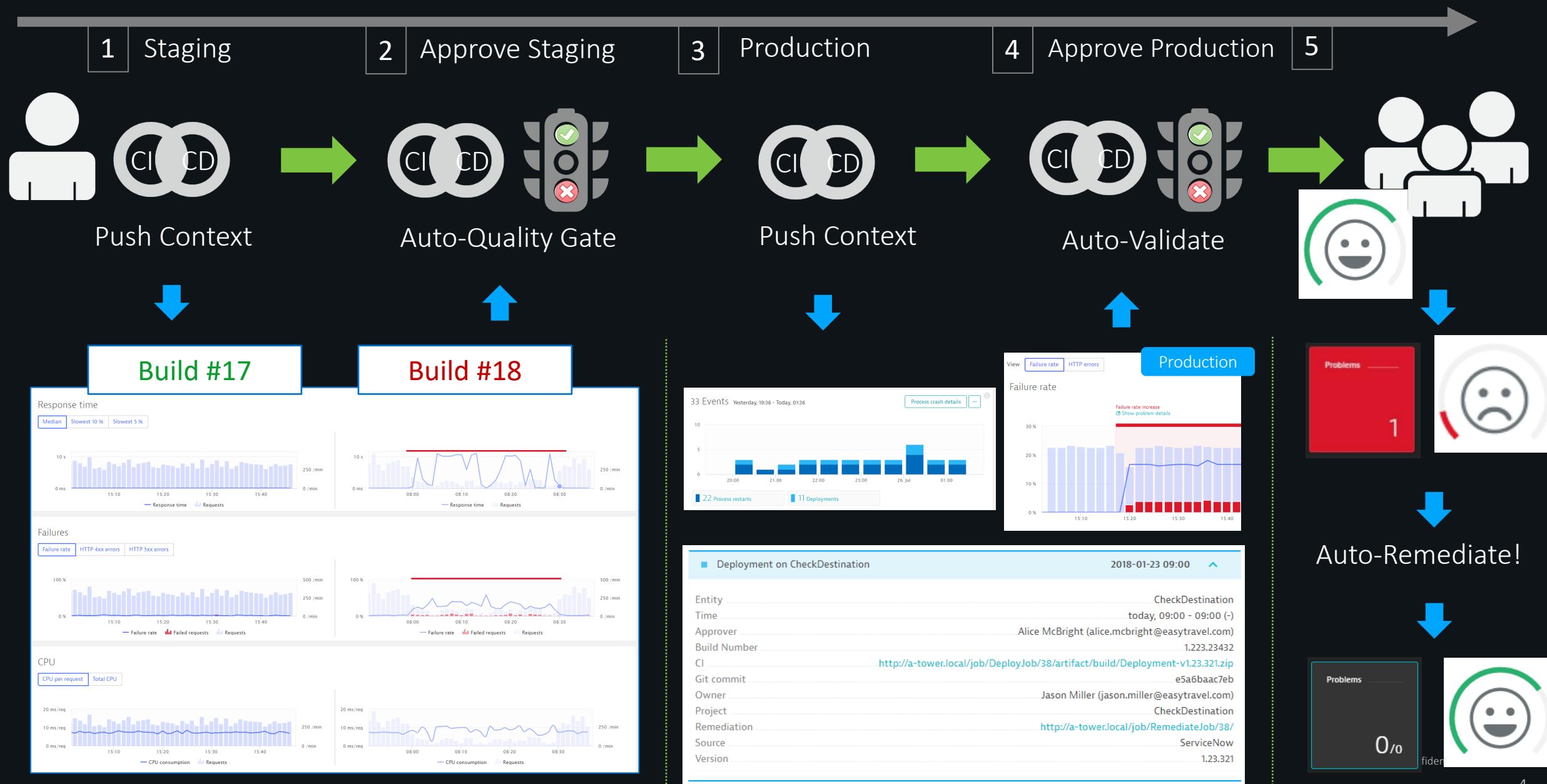


Review

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

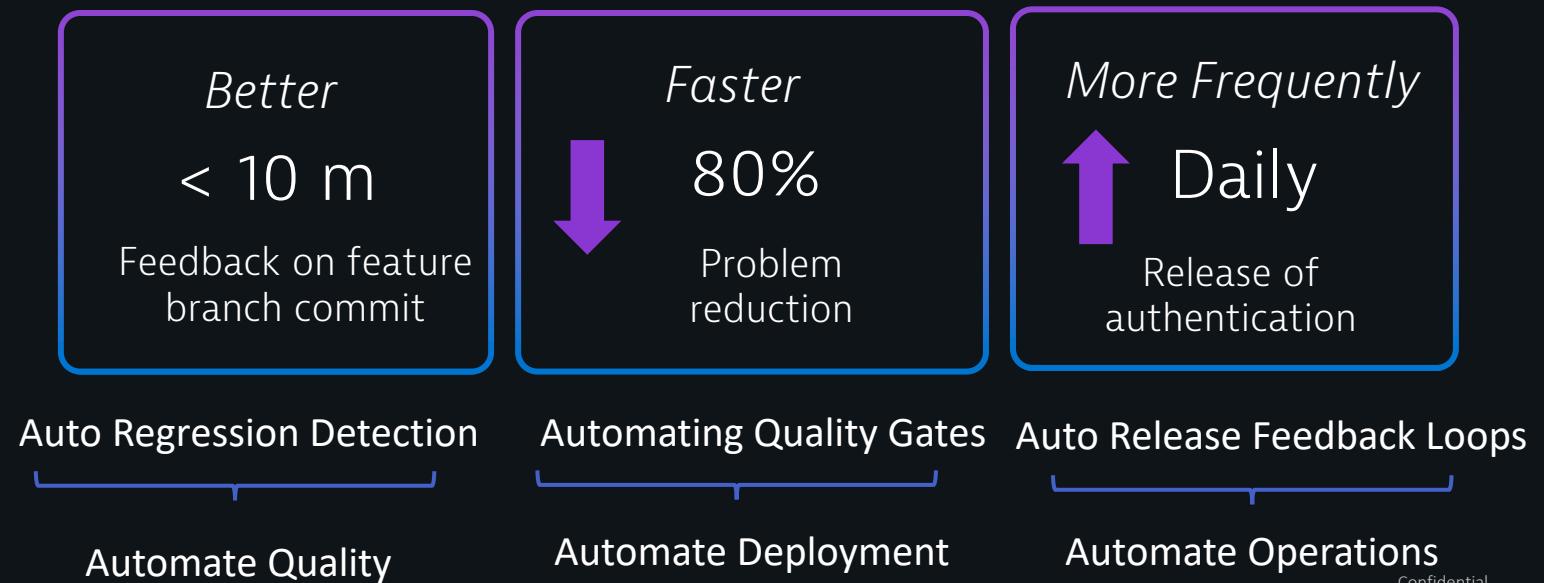
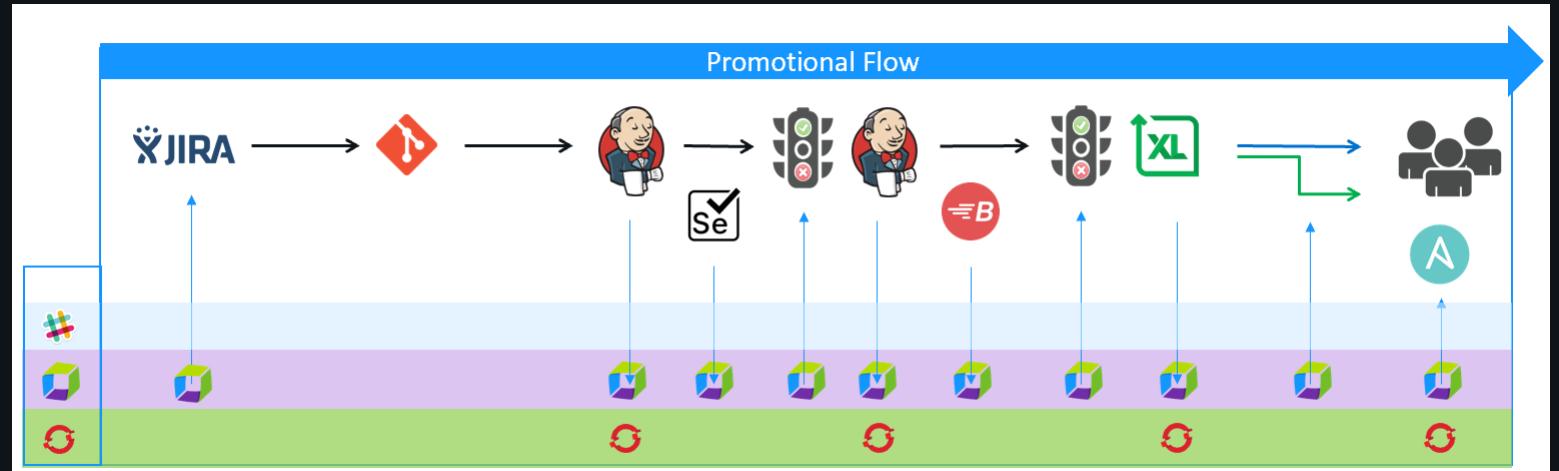


AIOps / ACM

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



Mick Miller, Sr DevOps Architect

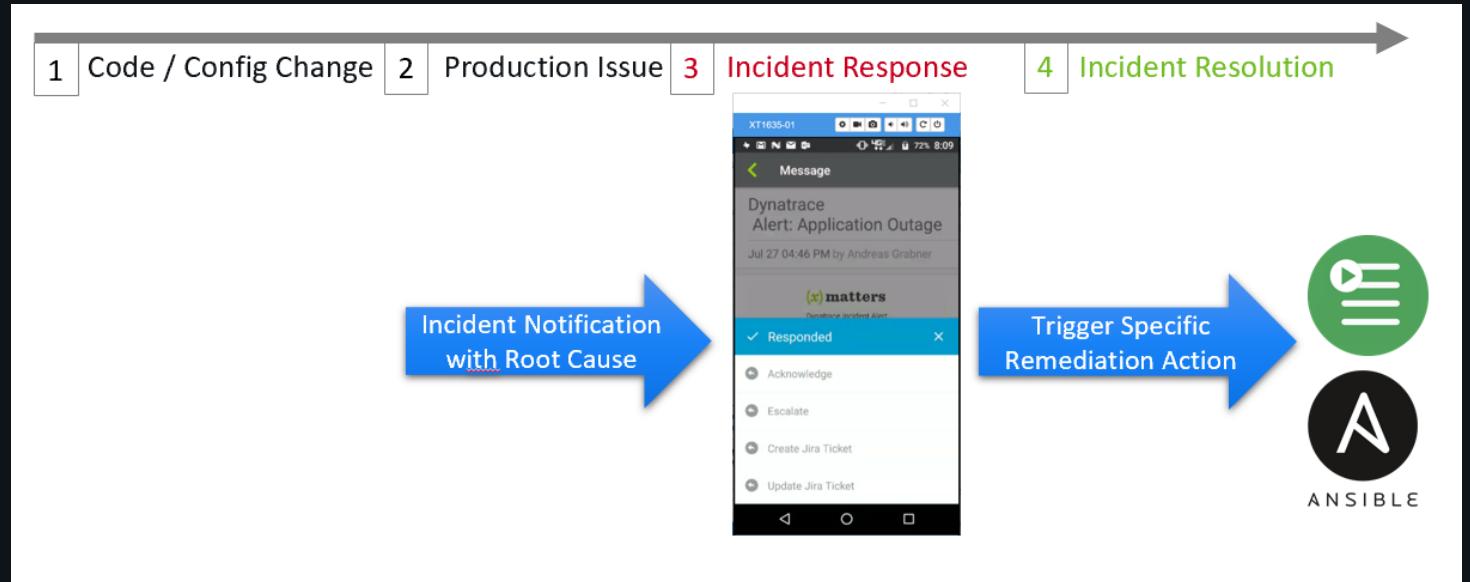


Auto-remediation

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



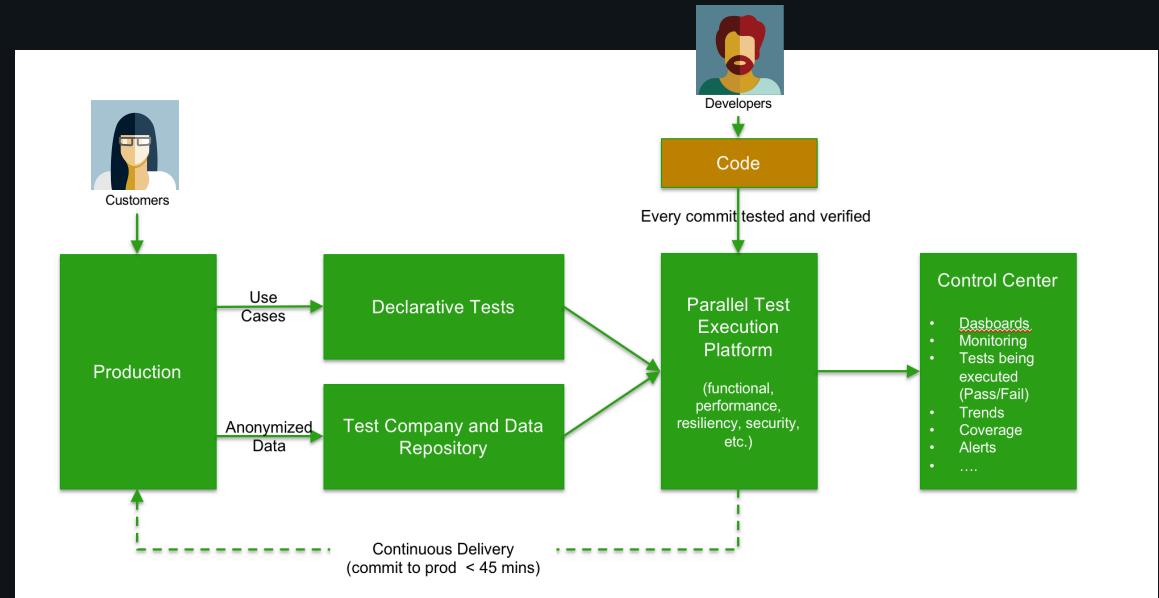
- Building resilience – making better, resilient systems
- Smart auto-remediation speeds up MTTR reduces risk
- Prevent in CI/CD versus REPAIR in production with quality gates
- Auto-remediation-as-code (resilience in the code pack) address root cause and not just symptom

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.



The Keptn logo features a blue icon composed of three horizontal bars of increasing length from left to right, followed by the word "keptn" in a lowercase sans-serif font.

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