

Unbreakable Azure Devops Pipeline using Dynatrace

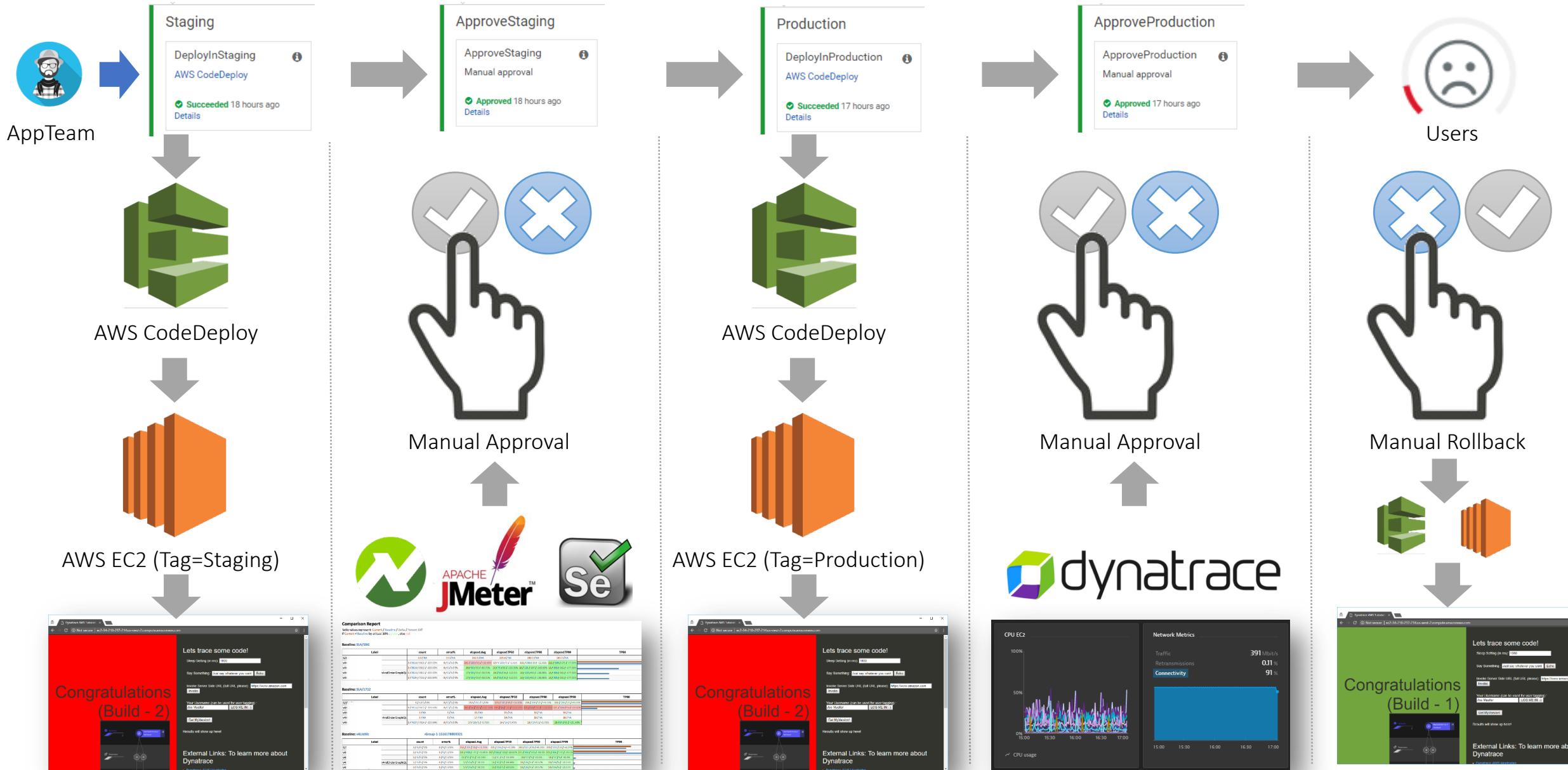
Safia Habib

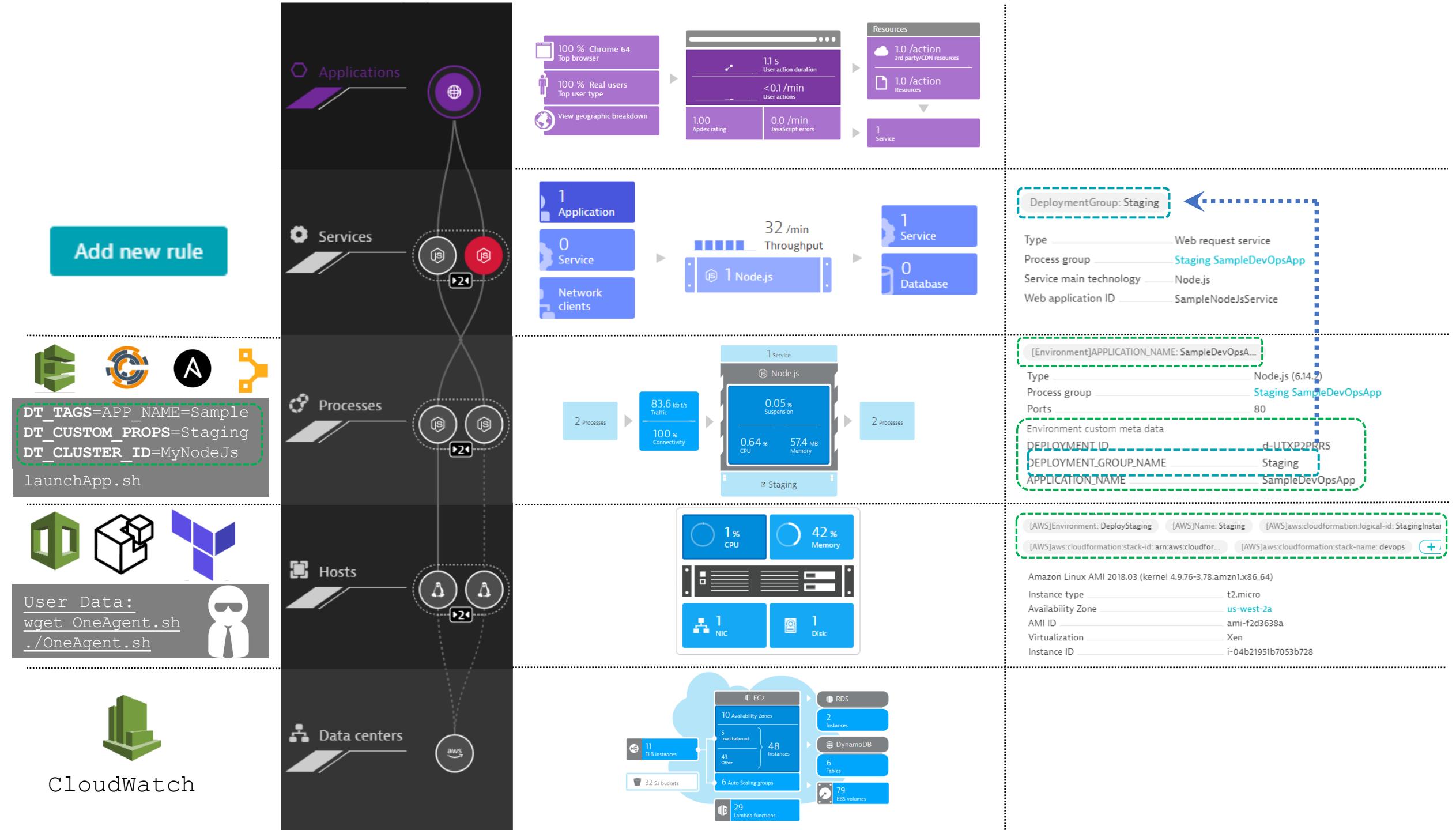
Pre Requisites

- Download the Source code from [here](#).
- Create Azure Devops account from <https://aka.ms/SignupAzureDevOps>
- Create Azure Portal account from <https://azure.microsoft.com/free/>
- A Dynatrace account, get a SaaS free trial here: <https://www.dynatrace.com/trial>
- Create a cheat sheet in a text editor (name : Azure_Devops_values) with the following:
 - DT_TENANT:
 - DT_TENANT_TOKEN
 - DT_API_TOKEN
 - URL to Monspec File:
 - URL to PipelineInfo File
 - PAT:
 - URL to UnbreakablePipelineGate function:
 - Token for UnbreakablePipelineGate function:
 - URL to SelfHealing function:

THE BIG PICTURE

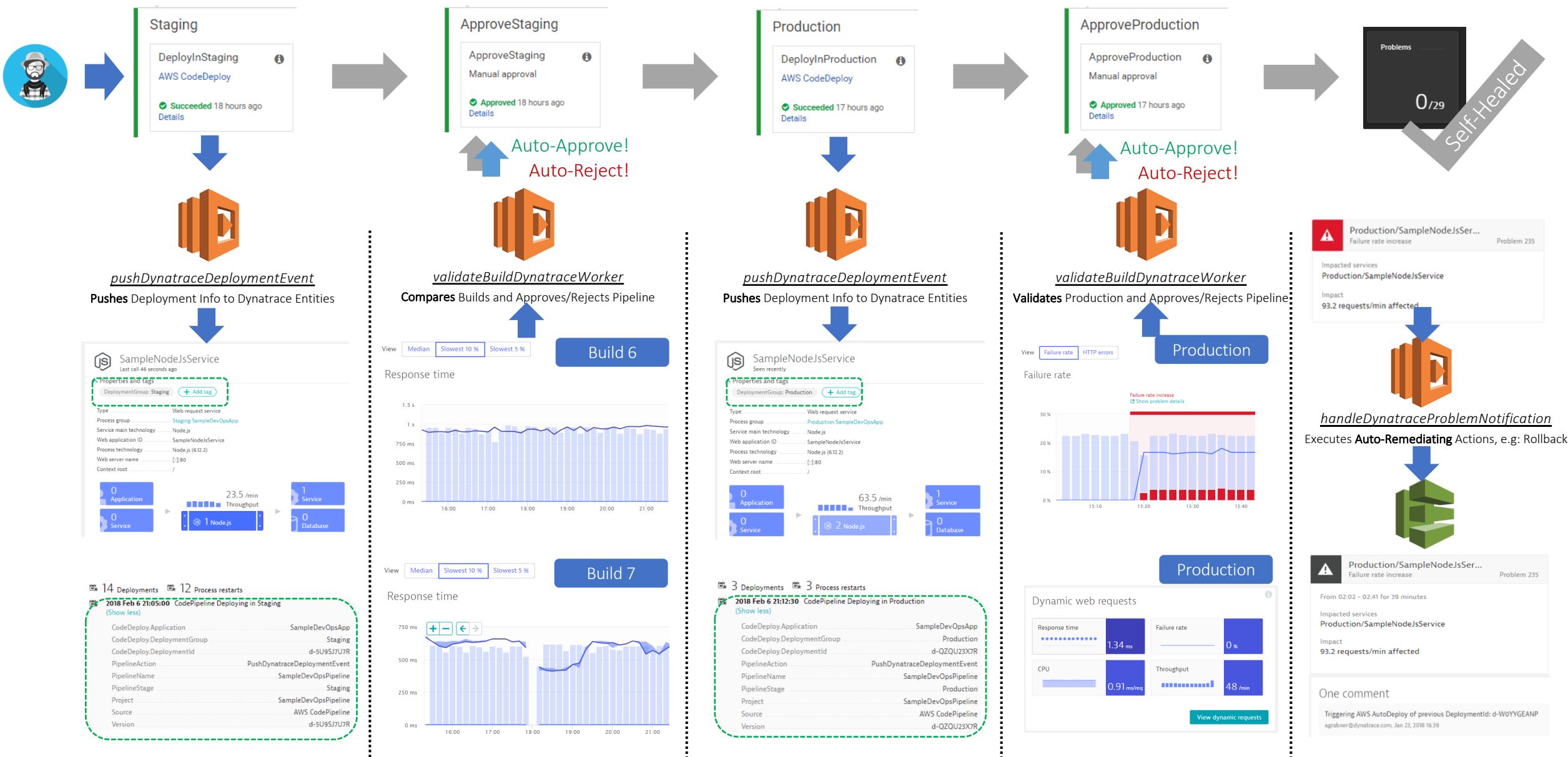
What we are going to do today!







<https://github.com/Dynatrace/AWSDevOpsTutorial>



For Azure and Azure DevOps

- Azure Devops provides multiple development collaboration tools. We will be using:
 - Azure Repos: Source control repository for the test application
 - Azure Pipelines: Creating CI/CD pipelines for deploying the test application
 - Azure Devops Dynatrace Extension: Deployed in Azure Marketplace.
 - Push Deployment to Dynatrace Task
 - Release Gate
- Azure Functions :
 - Self Healing Function: This function will redeploy a good build in the event that the bad build has made it to production upon receiving a Performance issue from Dynatrace.
 - Unbreakable Release gate Function – This is called post deployment from Staging to validate the build
- Azure Storage account: Used for storing the monspec file and the pipelineinfo files.
- Azure Proxy: Dynatrace Proxy: Utility with Dynatrace cli installed for comparing the Performance information with the monspec file.

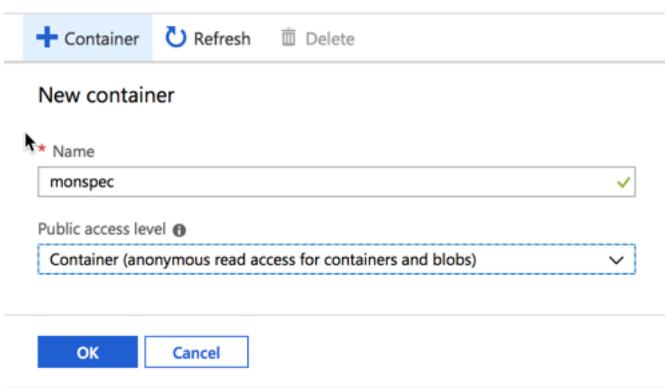
More on the Azure DevOps Dynatrace extension

- The Dynatrace Extension consists of two components:
 - Push Deployment Event Task:
 - The task pushes a Deployment event to Dynatrace when a new version of the application is deployed.
 - It uses the Dynatrace events API to publish the details of the Deployment.
 - The event requires URL to the Dynatrace Tenant, API token, Tag Name and Context (Note the Tag needs to be available in Dynatrace for the event to show up).
 - Unbreakable Release Gate:
 - The quality gate is executed to test the quality of the current build in Staging against a Monspec file.
 - In the case that there are no violations the gate succeeds and pushes the code to Production.
 - In case there are reported Violations then the Release Pipeline fails and the code is not deployed to Production.

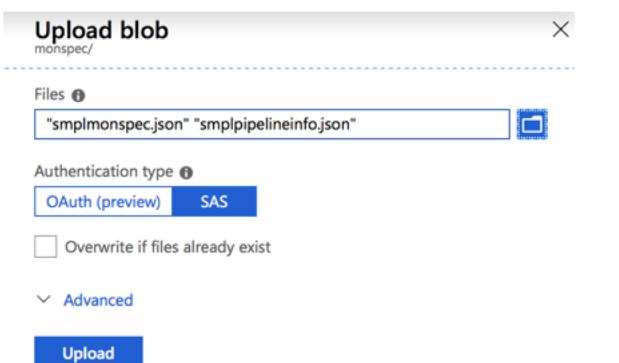
Step by Step

Step 1 : Storage Account

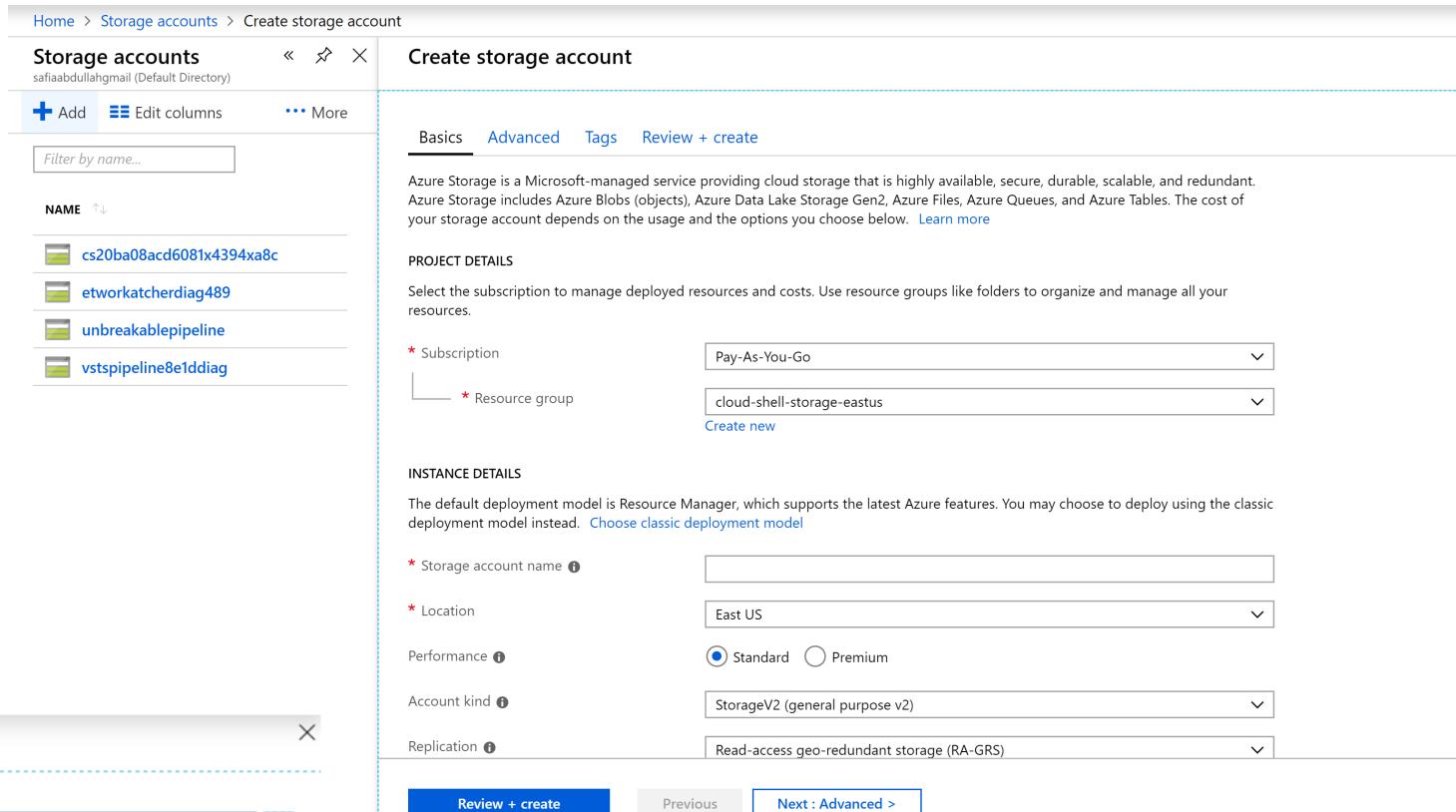
- Login to Azure Portal
- Create Storage Account
- Upload the Monspec folder from the source code to a new Blob container



The screenshot shows the 'Storage accounts' blade in the Azure portal. It lists several existing storage accounts: cs20ba08acd6081x4394xa8c, etworkatcherdiag489, unbreakablepipeline, and vstspipeline8e1ddiag. Below the list, there are buttons for '+ Container', 'Refresh', and 'Delete'. A 'New container' section is present, with a 'Name' field containing 'monspec' and a 'Public access level' dropdown set to 'Container (anonymous read access for containers and blobs)'. At the bottom are 'OK' and 'Cancel' buttons.



The screenshot shows the 'Upload blob' dialog box. In the 'Files' input field, two files are listed: "smplmonspec.json" and "smplpipelineinfo.json". The 'Authentication type' dropdown is set to 'SAS'. There is an unchecked checkbox for 'Overwrite if files already exist'. At the bottom are 'Advanced' and 'Upload' buttons.



The screenshot shows the 'Create storage account' blade. The 'Basics' tab is selected. It includes sections for 'PROJECT DETAILS' (Subscription: Pay-As-You-Go, Resource group: cloud-shell-storage-eastus), 'INSTANCE DETAILS' (Storage account name: [empty], Location: East US, Performance: Standard, Account kind: StorageV2, Replication: Read-access geo-redundant storage (RA-GRS)), and navigation buttons 'Review + create', 'Previous', and 'Next : Advanced >'.

The screenshot shows the Azure Storage Explorer interface. On the left, the 'monspec' container is selected under 'Storage accounts > unbreakablepipeline - Blobs > monspec'. The 'Overview' tab is active. In the center, the 'smplmonspec.json' blob is selected under 'Blobs' in the 'monspec' container. The 'Properties' tab is active, displaying the following details:

PROPERTY	VALUE
URL	https://unbreakablepipeline.bl...
LAST MODIFIED	10/30/2018, 2:13:06 PM
CREATION TIME	10/30/2018, 2:13:06 PM
TYPE	Block blob
SIZE	6.8 KiB
SERVER ENCRYPTED	true
ETAG	0x8D63E9357603890
CONTENT-MD5	9o7FusZZsarwRtWBBmdU0Q==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	-
COPY COMPLETION TIME	-

Below the properties, there is a button labeled 'Undelete all snapshots' and an 'Access Tier' section with a dropdown menu set to 'Hot (Inferred)'.

In the Azure_Devops_Values
Notepad copy the URL's of
the Monspec file and the
Pipeline Info File

Step 2: Create Azure DevOps Organization and Project

- Login to <https://dev.azure.com>
- Create a New Organization
- Create a New Project

Create new project x

Project name *

 ✓

Description

Visibility



Public ⓘ
Anyone on the internet
can view the project.
Certain features like
TFVC are not supported.



Private
Only people you give
access to will be able to
view this project.

Public projects are disabled for your organization. You can turn on public visibility with [organization policies](#).

▼ Advanced

Step 3: Setup Azure Repos

- In your project, go to Repos
- Import a Git Repository
- URL to clone is:
<https://github.com/safia-habib/UnbreakablePipelineSource>

Import a Git repository ×



Source type
Git

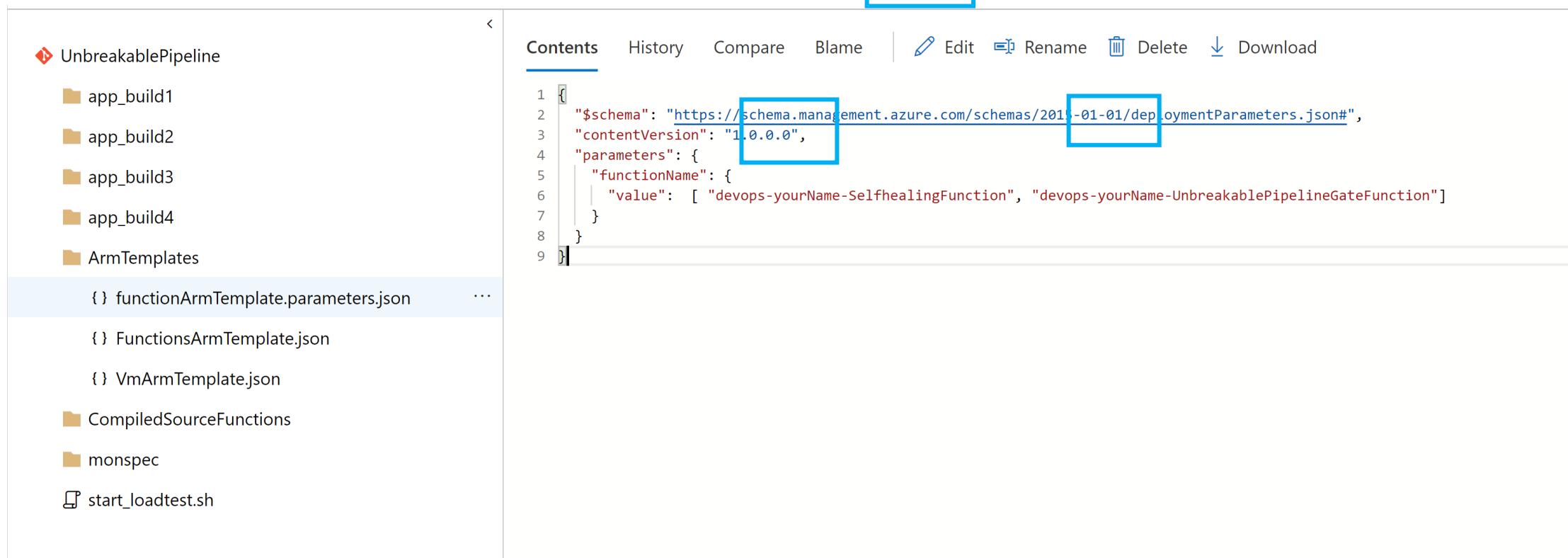
Clone URL *

Requires authorization

Name *

Import Close

- In the imported code navigate to ArmTemplates/functionarmTemplate.parameters.json
- Replace <your Name> With your First + Last name, for example : stevecaron
- Commit the changes once done



```
1 {  
2   "$schema": "https://schema.management.azure.com/schemas/2018-01-01/deploymentParameters.json#",  
3   "contentVersion": "1.0.0.0",  
4   "parameters": {  
5     "functionName": {  
6       "value": [ "devops-yourName-SelfhealingFunction", "devops-yourName-UnbreakablePipelineGateFunction" ]  
7     }  
8   }  
9 }
```

Step 4: Create a Personal Access Token

- Create a PAT and Copy Value to the Notepad

unbreakablepipeline / vstspipeline / Overview / Summary

The screenshot shows the Microsoft VSTS (now Azure DevOps) interface for managing personal access tokens (PATs). A blue box highlights the 'Personal access tokens' section in the 'User settings' sidebar. Another blue box highlights the 'New Token' button in the 'Personal Access Tokens' list. A large blue arrow points from the 'Security' menu item in the top navigation bar down to the 'Edit a personal access token' dialog, which is displayed on the right side of the screen.

Edit a personal access token

Name *
UnbreakablePipeline
Organization
unbreakablepipeline
Expiration (UTC)
Custom defined
Thu Nov 29 2018

Scopes
Full access

My profile
Security
Usage
Notification settings
Preview features
Theme

Project stats
Repos 0

User settings
General
Notifications
Usage
Security
Personal access tokens
Alternate credentials
SSH public keys

New Token
UnbreakablePipeline
Full access

Revoke
Edit
Re

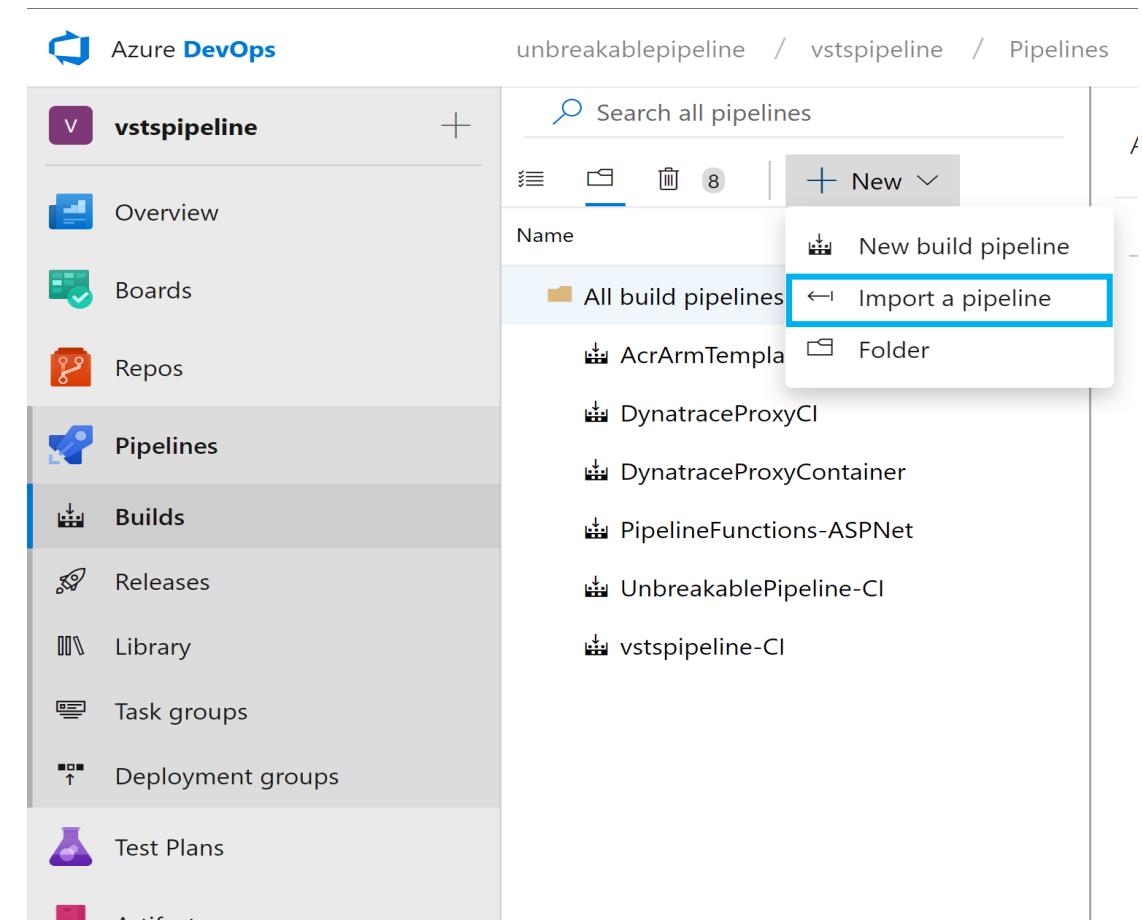
About this project
Help others to get on board!
Describe your project and make it easier for other people to understand it.
+ Add Project Description

vstspipeline

Step 5: Build Pipeline

- Go to Pipelines->Build
- Create a New Pipeline
- Import the Unbreakablepipeline-CI.json
- Current Limitation – Name of the Pipeline should match Unbreakablepipeline-CI so you might have to change the name assigned at the import (remove - import)

We have noticed that for new users there is sometimes no option to Import pipeline
In this case create a Dummy Pipeline With any name
To get to the below view and then import the Build pipeline



Result

unbreakablepipeline / vstspipeline / Pipelines

Search  ⌂  

… > UnbreakablePipeline-Cl

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue …

Pipeline Build pipeline

Get sources  UnbreakablePipeline  master

Agent job 1  +

Downloading the app for the Build  Bash

↑ Publish Artifact: app Publish Build Artifacts

↑ Publish Artifact: ArmTemplates Publish Build Artifacts

↑ Publish Artifact: FunctionsCode Publish Build Artifacts

Select a source

 Azure Repos Git  GitHub  GitHub Enterprise  Subversion

 Bitbucket Cloud  External Git

Team project  vstspipeline

Repository  UnbreakablePipeline

Default branch for manual and scheduled builds  master

Clean ⓘ

Ensure this matches the Repo Name You created in Step 2

Step 6: Run the Build Pipeline

- Queue a Build Pipeline

… > UnbreakablePipeline-CI

The screenshot shows the Azure Pipelines interface for a pipeline named 'UnbreakablePipeline-CI'. The top navigation bar includes 'Tasks', 'Variables', 'Triggers', 'Options', 'Retention', 'History', 'Save & queue', 'Discard', 'Summary', **Queue** (which is highlighted with a blue box), and an ellipsis button. On the right, there's a 'View YAML' button and a refresh icon. The main area is divided into two sections: 'Pipeline' (Build pipeline) and 'Agent job 1' (Run on agent). The 'Pipeline' section contains a 'Get sources' task (UnbreakablePipeline, master branch) and three 'Publish Artifact' tasks: 'Downloading the app for the Build' (Bash), 'Publish Artifact: app' (Publish Build Artifacts), 'Publish Artifact: ArmTemplates' (Publish Build Artifacts), and 'Publish Artifact: FunctionsCode' (Publish Build Artifacts). The 'Agent job 1' section shows the 'Agent pool' dropdown set to 'Hosted VS2017'. Below the agent job, there's a 'Parameters' section with a note about creating pipeline parameters and a 'Learn more' link.

Pipeline
Build pipeline

Get sources
UnbreakablePipeline master

Agent job 1
Run on agent

+

#! Downloading the app for the Build
Bash

↑ Publish Artifact: app
Publish Build Artifacts

↑ Publish Artifact: ArmTemplates
Publish Build Artifacts

↑ Publish Artifact: FunctionsCode
Publish Build Artifacts

Name *
UnbreakablePipeline-CI

Agent pool * ⓘ | Pool information | Manage ↗
Hosted VS2017

Parameters ⓘ
This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.
Learn more ↗

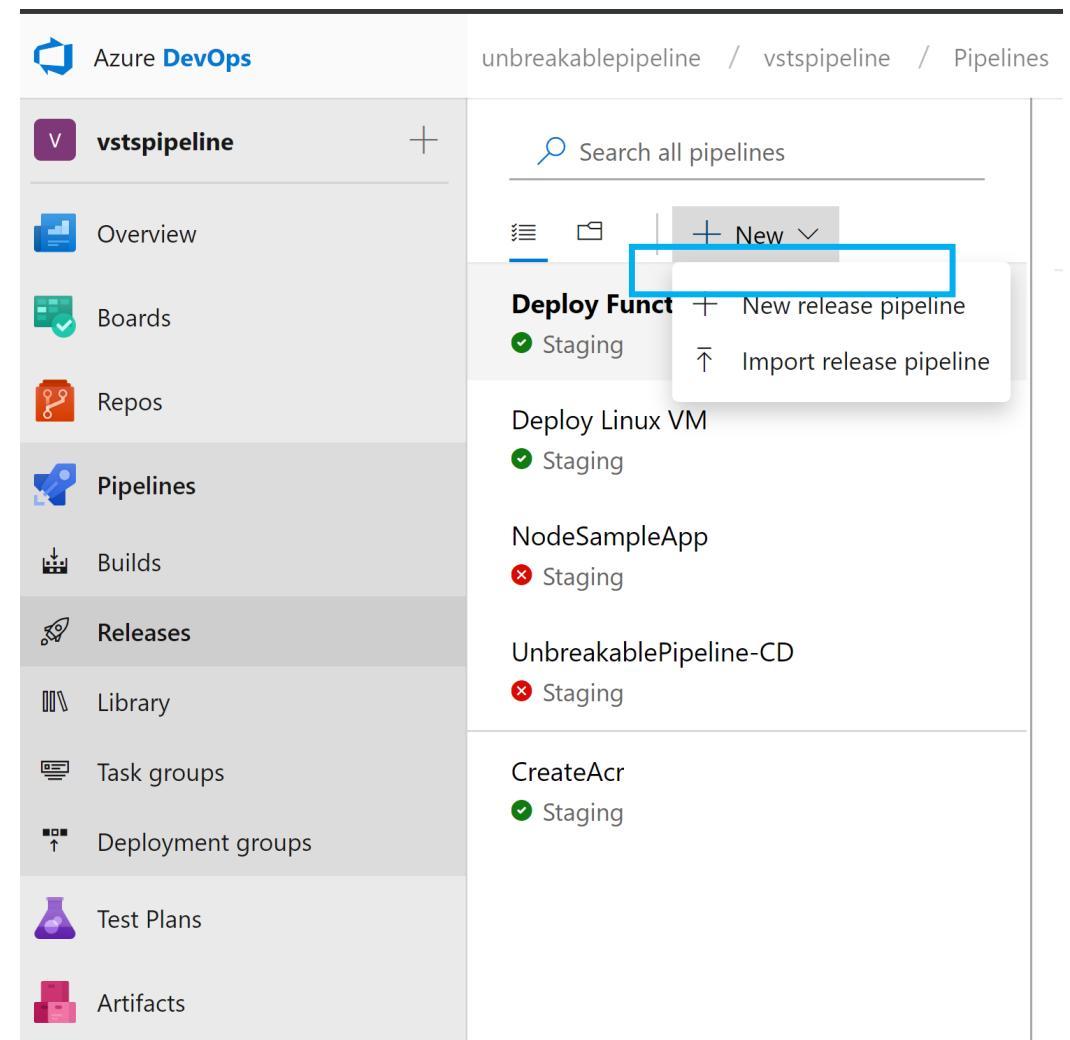
Step 7: Release Pipeline

- Go to Pipelines Release
- Import the CreateUtilities.json

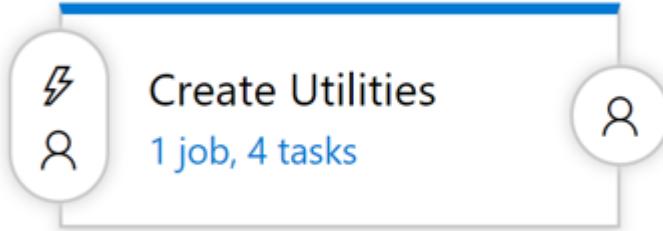
We have noticed that for new users there is sometimes no option to Import pipeline

In this case create a Dummy Pipeline With any name

To get to the below view and then import the Build pipeline



Release Stage



Create utilities

Creates the following in Azure:

- Self Healing Function
- Unbreakable Pipeline Gate Function
- For today we will just deploy the app to the Hosted agent / factory VM available provided by Azure Devops
- Staging Linux VM (Excluded for this Workshop)
- Production Linux VM (Excluded for this workshop)

Step 8: Deploy time

- First run the Create Utilities Stage

The screenshot shows the Azure DevOps Pipeline interface for the 'Create Utilities' stage of the 'UnbreakablePipeline-CD' release definition. The pipeline has completed successfully.

Pipeline Navigation: UnbreakablePipeline-CD > Release-110 > Create Utilities (Succeeded)

Logs Tab: Selected tab, showing deployment logs.

Deployment process: Succeeded

Agent job: Agent job (Succeeded)

Agent job Details:

- Pool: Hosted VS2017 · Agent: Hosted Agent
- Started: 11/6/2018, 3:40:07 PM
- Duration: 3m 16s

Task	Status	Duration
Initialize Agent	succeeded	4s
Initialize job	succeeded	3s
Download artifact - _UnbreakablePipeline-Cl - app	succeeded	2s
Download artifact - _UnbreakablePipeline-Cl - ArmTemplates	succeeded	1s
Download artifact - _UnbreakablePipeline-Cl - FunctionsCode	succeeded	3s
Create the Linux VM's in Azure	succeeded	3m 6s

Step 9: Functions URL and Key

- Go to Azure Portal and copy the Self Healing Function URL Azure_Devops_values notepad.

The screenshot shows the Azure Portal interface. On the left, the navigation menu is visible with various services listed. The 'Function Apps' item is selected and highlighted with a blue box. The main content area shows the 'Overview' tab for a function app named 'devops-yourName-SelfhealingFunction'. The URL of the function app is displayed in the 'URL' section: <https://devops-yourname-selfhealingfunction.azurewebsites.net>. This URL is also highlighted with a blue box.

Home > devops-yourName-SelfhealingFunction

devops-yourName-SelfhealingFunction

Function Apps

Search

All subscriptions

Overview Platform features

Status: Running Subscription: Pay-As-You-Go Resource group: VstsResourceGroup-vstspipeline

Availability: Not applicable Subscription ID: 0ba08acd-6081-4394-a8c4-ce2ffd1e238f Location: Central US

URL: <https://devops-yourname-selfhealingfunction.azurewebsites.net>

App Service plan / pricing tier: UnbreakableFunctionApps (Free)

Configured features

- Function app settings
- Application settings
- Deployment options configured with VSTS RM

Step 9: Functions URL and Key

- Get the Unbreakablepipeline function URL and Key – Copy to Notepad

The screenshot shows the Azure Functions portal interface for the function app "devops-yourName-UnbreakablePipelineGateFunction - ProcessUnbreakableGate". The left sidebar is collapsed, and the main content area displays the function keys.

Function Keys:

NAME	VALUE	ACTIONS
default	Click to show	Copy Renew Revoke

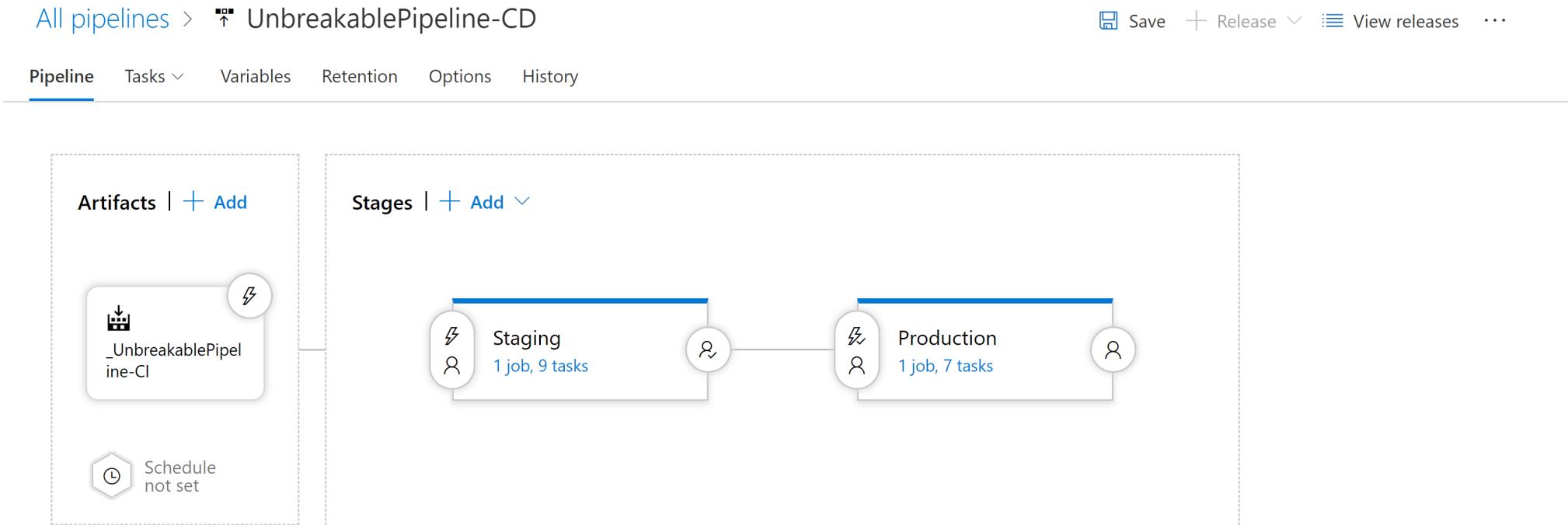
Host Keys (All functions):

NAME	VALUE	ACTIONS
_master	Click to show	Copy Renew
default	Click to show	Copy Renew Revoke

Two specific actions are highlighted with red boxes: the "Copy" link under the "default" function key in the Function Keys section, and the "Copy" link under the "_master" host key in the Host Keys section.

App Pipeline

- Import UnbreakablePipeline-CD.json into New Release Pipeline



App Release Pipeline



Staging

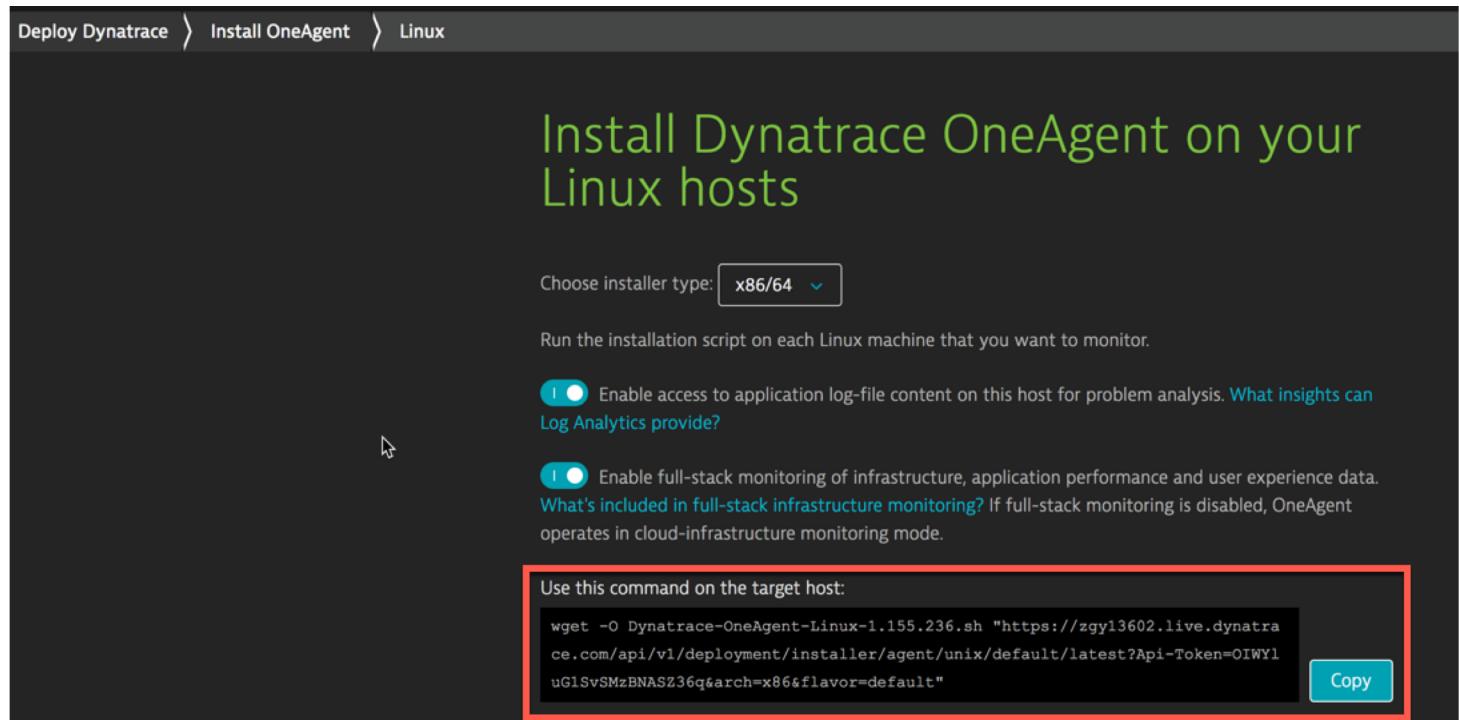
- Deploys the test app on Staging VM
- Starts Loadtest against the test application
- Pushes Deployment Event to Dynatrace

Production

- Deploys the test app on Staging VM
- Starts Loadtest against the test application
- Pushes Deployment Event to Dynatrace

Step 10 : Dynatrace Account Info

- Menu->Deploy
Dynatrace->Install
OneAgent->Linux
- Copy the wget
command
- Copy the tenant URL
and the installation
token on your notepad
- DT_TENANT:
<https://<tenant>.live.Dynatrace.com>
- DT_TENANT_TOKEN:
Value of Tenant token



Deploy Dynatrace > Install OneAgent > Linux

Install Dynatrace OneAgent on your Linux hosts

Choose installer type: x86/64

Run the installation script on each Linux machine that you want to monitor.

Enable access to application log-file content on this host for problem analysis. [What insights can Log Analytics provide?](#)

Enable full-stack monitoring of infrastructure, application performance and user experience data. [What's included in full-stack infrastructure monitoring?](#) If full-stack monitoring is disabled, OneAgent operates in cloud-infrastructure monitoring mode.

Use this command on the target host:

```
wget -O Dynatrace-OneAgent-Linux-1.155.236.sh "https://zgy13602.live.dynatrace.com/api/v1/deployment/installer/agent/unix/default/latest?Api-Token=OIWYluG1SvSMzBNASZ36q&arch=x86&flavor=default"
```

[Copy](#)

```
wget -O Dynatrace-OneAgent-Linux-1.155.236.sh 'https://zgy13602.live.dynatrace.com/api/v1/deployment/installer/agent/unix/default/latest?Api-Token=OIWYluG1SvSMzBNASZ36q&arch=x86&flavor=default'
```

Step 11 : Dynatrace API Token

- Menu->Settings->
Integration->
Dynatrace API
- Generate Token
- Copy the token value

Generated token

eBWLBwRMRQq8zWIJKTfH1

Copy

Token name

Use the switches below to define the access scope of your Dynatrace API token.

- Access problem and event feed, metrics, topology and RUM JavaScript tag management
- Access logs
- Configure maintenance windows
- Send external synthetic data to Dynatrace
- User session query language
- Anonymize user session data for [data privacy](#) reasons
- Log import

Step 12: Add Auto Tagging rule to Dynatrace

The screenshot shows the Dynatrace interface for managing automatically applied tags. A blue arrow labeled '#1' points to the 'Automatically applied tags' section in the left sidebar. Another blue arrow labeled '#2 – Call the Tag “DeploymentGroup”' points to the 'DeploymentGroup' title. A third blue arrow labeled '#3 – For Services' points to the 'Rule applies to' dropdown set to 'Services'. A fourth blue arrow labeled '#4 – Tag Value comes from our Process Environment Variable' points to the 'Optional tag value' field containing '(ProcessGroup:Environment:DEPLOYMENT_GROUP_NAME)'. A fifth blue arrow labeled '#5 – Only apply rule if env variable exists' points to the 'Conditions' section where 'DEPLOYMENT_GROUP_NAME (Environment)' is selected with the 'exists' operator.

Settings > Tags > Automatically applied tags > DeploymentGroup

DeploymentGroup #2 – Call the Tag “DeploymentGroup”

This custom tag is applied to all services, process groups, or hosts that match one or more of the rules defined below. Matching entities are listed at the bottom of this page.

It can take up to a minute for rules to detect newly added entities and changes to existing entities.

Add new rule

Rule _____

Services on process groups where DEPLOYMENT_GROUP_NAME (Environment) exists #4 – Tag Value comes from our Process Environment Variable

Rule applies to #3 – For Services

Services of the following technologies and topology

Optional tag value

(ProcessGroup:Environment:DEPLOYMENT_GROUP_NAME)

Available placeholders

(aws Auto Scaling Group Name)

(awsRelationalDatabaseService:Endpoint)

(awsRelationalDatabaseService:InstanceClass)

(awsRelationalDatabaseService:Name)

(AzureRegion:Name)

(AzureScaleSet:Name)

(AzureVm:Name)

(CustomDevice:DetectedName)

(CustomDevice:DnsName)

(CustomDevice:IpAddress)

(CustomDevice:Port)

(DockerContainerGroupInstance:ContainerName)

(DockerContainerGroupInstance:FullImageName)

(DockerContainerGroupInstance:ImageVersion)

More...

Conditions: #5 – Only apply rule if env variable exists

All conditions listed below must be met for this rule to be applied.

DEPLOYMENT_GROUP_NAME (Environment) exists

Add condition

Apply tag to underlying process groups of matching services

Apply tag to underlying hosts of matching services

Step 13: Create Alerting Profile for Production

Settings > Alerting > Alerting profiles > Edit profile

ProductionService

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

[Create alerting rule](#)

Rule	Delete	Edit
Error alert (Immediate; DeploymentGroup: Production)	X	^
Slowdown alert (After 30 mins; DeploymentGroup: Production)	X	^
Custom alert (Immediate; DeploymentGroup: Production)	X	^

Problem severity level: Error

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: DeploymentGroup:Production

Remove filter

Create tag filter

Cancel Save

Reset to default

Settings

Monitoring

- Setup and overview
- Monitoring overview
- Process group detection
- Monitored technologies

Web and mobile monitoring

- Global settings and configuration

Cloud and virtualization

- Connect vCenter or Amazon account

Server-side service monitoring

- Manage & customize service monitoring

Log analytics

- Customize detection of log-based events

Anomaly detection

- Configure detection sensitivity

Alerting

- Configure alerting settings

Alerting profiles

Integration

- Integrate Dynatrace with 3rd party systems...

Tags

Step 14: Add SelfHealing Function URL Webhook Integration

- To enable auto remediation add the Self Healing Function URL to Dynatrace after creating the alerting profile
- WebhookURL:
SelfHealingFunction URL +
/api/ProcessAlert?name=
<YourName>
- Don't forget to select the Production alerting profile (at the bottom)

The screenshot shows the 'Edit custom integration' page in the Dynatrace interface. The navigation path is: Settings > Integration > Problem notifications > Edit custom integration. The left sidebar lists various monitoring categories: Monitoring, Monitored technologies, Monitoring overview, Host naming, Process groups, Detection and naming, Web & mobile monitoring, Real user & synthetic monitoring, Cloud and virtualization, Connect vCenter, Azure, Cloud Foundry ..., Server-side service monitoring, Manage & customize service monitoring, Log Analytics, Set up management of logs, Anomaly detection, Configure detection sensitivity, and Alerting. The main configuration area includes:

- Name:** VSTSSelfHealingHandler
- Webhook URL:** A field with a blue placeholder bar.
- Accept any SSL certificate (Self signed or invalid)**
- Additional HTTP headers:** Buttons for '+ Add header' and 'Create basic authorization header'.
- Custom payload:** A code editor containing a JSON template:

```
{  
  "State": "{State}",  
  "ProblemID": "{ProblemID}",  
  "ProblemTitle": "{ProblemTitle}",  
  "PID": "{PID}",  
  "ImpactedEntities": {ImpactedEntities}  
}
```
- Available placeholders:** A section explaining the placeholder {ImpactedEntities}.

Step 14 – Release Pipeline Configuration

- After Importing the release pipeline go to each stage
- For Each Task in each stage you will have to select the Azure Subscription, Resource Group, Location
- Add the following values for the Variables in the Variables Section

All pipelines > UnbreakablePipeline-CD

Save + Release View releases ...

Pipeline Tasks Variables Retention Options History

Pipeline variables Variable groups Predefined variables

Filter by keywords Scope X List Grid

Name	Value
ApplicationName	SampleDevOpsApp
DeploymentId	56789
DT_API_TOKEN	[REDACTED]
DT_TENANT	[REDACTED]
DT_TENANT_TOKEN	[REDACTED]
ServiceToCompare	SampleJsonService/StagingToProduction
SimulateBuildNumber	2

< >

Push Deployment Event Task

- Update the values in Push Deployment Event Task for Staging and Production

Dynatrace Push Deployment Event ⓘ

X Remove

Version 0.* ▾

Display name *

Dynatrace tenant token * ⓘ

Dynatrace tenant URL * ⓘ

Entity Type * ⓘ

Tag Context * ⓘ

Tag Name * ⓘ

Tag Value * ⓘ

Update Release gate

Dynatrace Unbreakable Pipeline Relea...	<input checked="" type="checkbox"/> Enabled	
Dynatrace Unbreakable Release Gate <small>i</small>		
Display name *	<input type="text"/> Dynatrace Unbreakable Pipeline Release Gate	
Unbreakable Gate Function Url *	<input type="text"/>	
Unbreakable Gate Function Key *	<input type="text"/>	
Monspec Url *	<input type="text"/>	
Pipeline Info Url *	<input type="text"/>	
Dynatrace Tenant Url *	<input type="text"/> \$(DT_TENANT)	
Dynatrace Token *	<input type="text"/> \$(DT_API_TOKEN)	
Dynatrace Proxy Url *	<input type="text"/> http://40.117.92.217:5000/api/DTCLIPProxy/MonspecPullRequest	
Service To Compare * <small>i</small>		
<input type="text"/> SampleJsonService/StagingToProduction		
Compare Window *	<input type="text"/> 5	
VSTS Plan Url *	<input type="text"/> \$(system.CollectionUri)	
VSTS Project Id *	<input type="text"/> \$(system.TeamProjectId)	
Hub Name *	<input type="text"/> \$(system.HostType)	
Plan Id *	<input type="text"/> \$(system.PlanId)	
Job Id *	<input type="text"/> \$(system.JobId)	
Timeline Id *	<input type="text"/> \$(system.TimelineId)	
Task Instance Id *	<input type="text"/> \$(system.TaskInstanceId)	
Authentication Token *	<input type="text"/> PAT Token	

Values for Release Gate

- Note: Unbreakable Gate Function URL : URL +
api/ProcessUnbreakableGate
- Dynatrace Proxy URL:
<http://40.117.92.217:5000/api/DTCLIPProxy/MonspecPullRequest>
- Service to Compare value: SampleJsonService/StagingToProduction
- Authentication Token : PAT Token

Create Release

- View Deployment in Dynatrace
- Validate Release Gates

Behind the Scenes: Deployment

Shift-Right: Version Information, Tags, Meta Data, ...

What happens in Release Pipeline Stages?

```
#1 Executes script shell to start the app:  
export DT_TAGS=APPLICATION_NAME=$APPLICATION_NAME  
export DT_CUSTOM_PROP="DEPLOYMENT_ID=$DEPLOYMENT_ID DEPLOYMENT_GROUP_NAME=$DEPLOYMENT_GROUP_NAME  
APPLICATION_NAME=$APPLICATION_NAME"  
export DT_CLUSTER_ID="$DEPLOYMENT_GROUP_NAME $APPLICATION_NAME"  
pm2 start app.js &> pm2start.log
```

The screenshot shows a Node.js application monitoring interface. On the left, there's a detailed properties and tags table for the 'Staging SampleDevOpsApp' environment. The table includes columns for Type (Node.js 6.12.2), Process group (Staging SampleDevOpsApp), Ports (80), Bitness (64-bit), Application version (6.12.2), Packages (SampleNodeJsService (0.0.1), debug (3.1.0), json-stringify-safe (5.0.1), ms (2.0.0), pm...), Command line args (node /usr/lib/node_modules/pm2/lib/ProcessContainerFork.js, node /usr/bin/npm i...), Dynatrace custom cluster ID (Staging SampleDevOpsApp), EXE name (node), EXE path (/usr/bin/node), Node.js application (SampleNodeJsService), Node.js script name (/usr/lib/node_modules/npm/bin/npm-cli.js), and Environment custom meta data (APPLICATION_NAME: SampleDevOpsApp, DEPLOYMENT_ID: d-QRHDKBQ0R, DEPLOYMENT_GROUP_NAME: Staging).

Three blue arrows point from the top of the slide to specific sections of the dashboard:

- #2 DT_CLUSTER_ID points to the 'DT_CLUSTER_ID' entry in the properties table.
- #2 DT_TAGS points to the 'DT_TAGS' entry in the properties table.
- #2 DT_CUSTOM_PROP points to the 'DT_CUSTOM_PROP' entry in the properties table.

In the center, a status bar says "No problems in last 72 hours". To the right, there's a service summary card for 'Node.js' showing 1 Service, 3 Processes, 90.6 kbit/s Traffic, 100% Connectivity, 0.06% Suspension, 0.75% CPU, and 53.9 MB Memory. Below the service summary is a 'Staging' section with a dropdown arrow.

Shift-Right: Tags, Deployments & Events

Release Automation



A



E

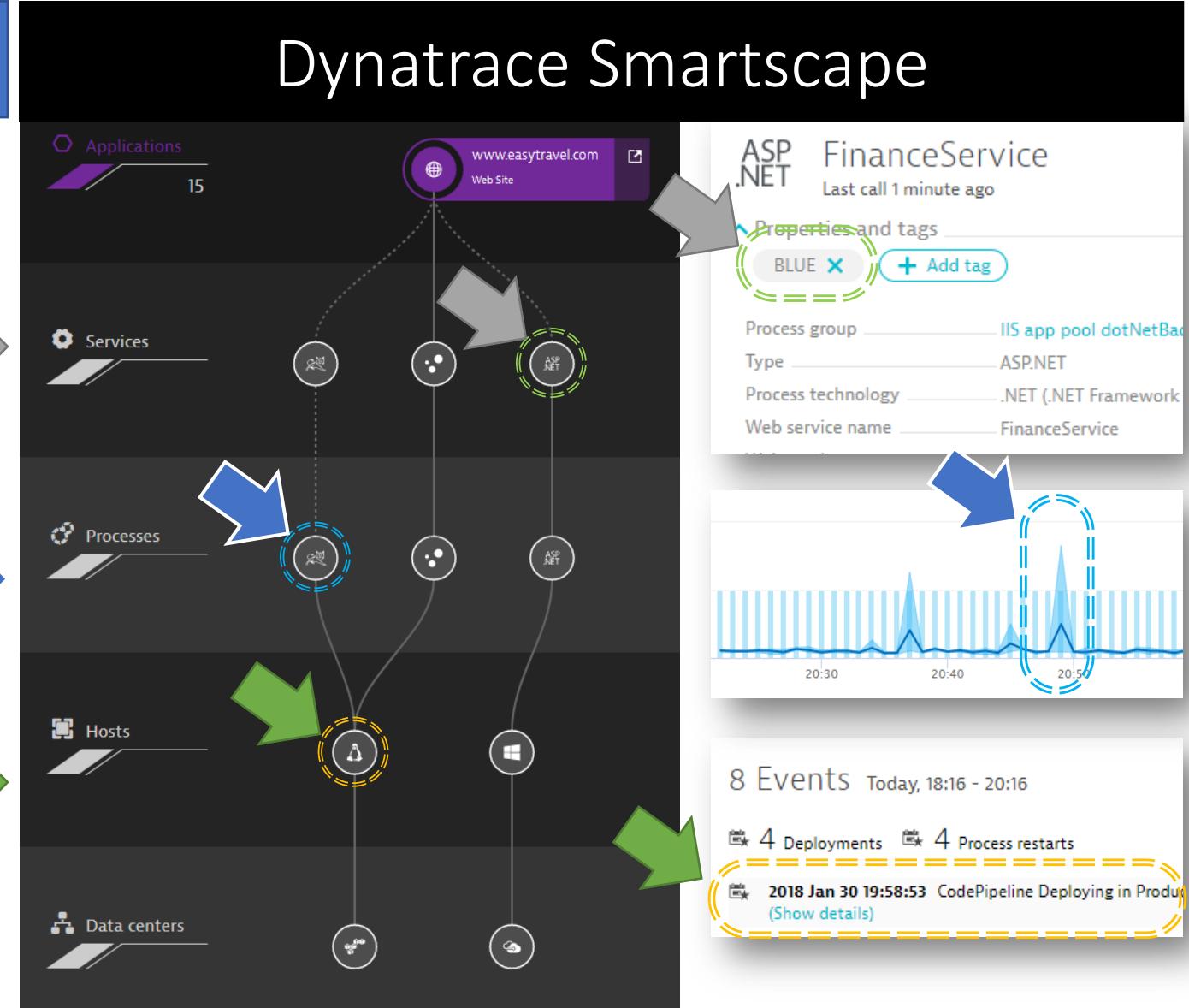
XL

Dynatrace Automation
API, CLI, Auto-Detection

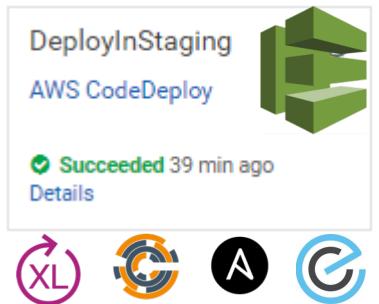
```
docker run -e DT_TAGS=BLUE  
dtcli tag srv CartServicev2 GREEN
```

```
dtcli evt push pg tomcat1  
desc=JVMMemIncr hint=+100MB
```

```
dtcli evt push host .*demo  
version=123 source={code_deploy}
```



@ Deployment: Passing Meta Data to launched processes

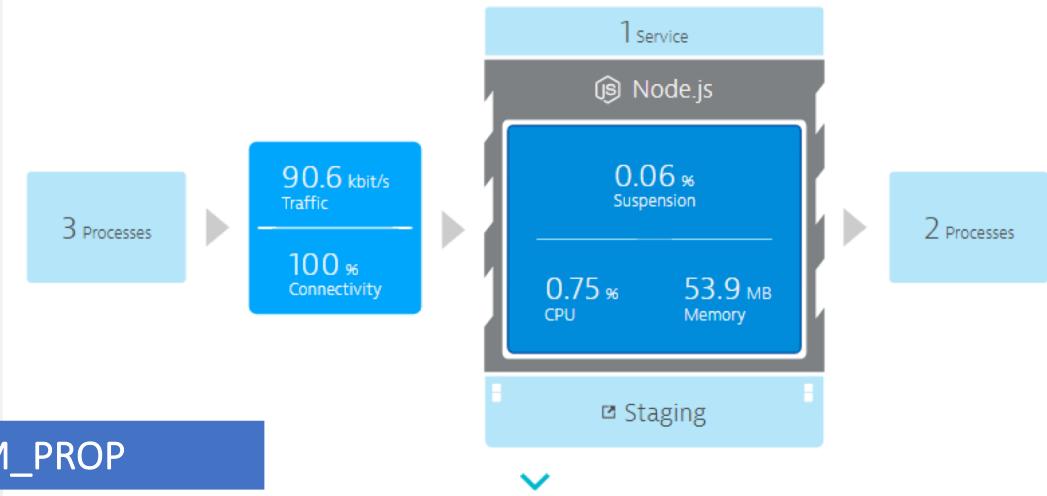


AWS CodeDeploy executes script during deployment:

```
export DT_TAGS=APPLICATION_NAME=$APPLICATION_NAME
export DT_CUSTOM_PROP="DEPLOYMENT_ID=$DEPLOYMENT_ID DEPLOYMENT_GROUP_NAME=$DEPLOYMENT_GROUP_NAME
APPLICATION_NAME=$APPLICATION_NAME"
export DT_CLUSTER_ID="$DEPLOYMENT_GROUP_NAME $APPLICATION_NAME"
pm2 start app.js &> pm2start.log
```



No problems in last 72 hours



@ Deployment: Converting Process Meta Data to Tags on Services!

Staging SampleDevOpsApp

Analyze process connections ...

Properties and tags

[Environment]APPLICATION_NAME: SampleDevOpsA...

Type	Node.js (6.12.2)
Process group	Staging SampleDevOpsApp
Ports	80
Bitness	64-bit
Application version	6.12.2
Packages	SampleNodeJsService (0.0.1), debug (3.1.0), json-stringify-safe (5.0.1), ms (2.0.0), pm...
Command line args	node /usr/lib/node_modules/pm2/lib/ProcessContainerFork.js, node /usr/bin/npm i...
Dynatrace custom cluster ID	Staging SampleDevOpsApp
EXE name	node
EXE path	/usr/bin/node
Node.js application	SampleNodeJsService
Node.js script name	/usr/lib/node_modules/npm/bin/npm-cli.js
Environment custom meta data	
APPLICATION_NAME	SampleDevOpsApp
DEPLOYMENT_ID	d-QRHDKB00P
DEPLOYMENT_GROUP_NAME	Staging

DeploymentGroup

Rule

Services on process groups where DEPLOYMENT_GROUP_NAME (Environment) exists

SampleNodeJsService
Seen recently

Properties and tags

DeploymentGroup: Staging

0 Application

0 Service

31.5 /min Throughput

1 Node.js

1 Service

0 Database

Processes and hosts

Process: Staging SampleDevOpsApp

Runs on: Staging

State: Available

Via Tagging Rule

@ Deployment: Converting Process Meta Data to Tags on Services!

Production SampleDevOpsApp

[Environment]APPLICATION_NAME: SampleDevOpsA...

Type Node.js (6.12.2)
Process group Production SampleDevOpsApp
Ports 80
Bitness 64-bit
Application version 6.12.2
Packages SampleNodeJsService (0.0.1), debug (3.1.0), json-stringify-safe (5.0.1), ms (2.0.0), pm...
Command line args node /usr/lib/node_modules/pm2/lib/ProcessContainerFork.js
Dynatrace custom cluster ID Production SampleDevOpsApp
EXE name node
EXE path /usr/bin/node
Node.js application SampleNodeJsService

Environment custom meta data
DEPLOYMENT_ID d-QZQU23X7R
APPLICATION_NAME SampleDevO...
DEPLOYMENT_GROUP_NAME Production

Via Tagging Rule

DeploymentGroup

Rule

Services on process groups where DEPLOYMENT_GROUP_NAME (Environment) exists

SampleNodeJsService
Seen recently

Properties and tags
DeploymentGroup: Production

0 Application
0 Service

67 /min Throughput

1 Node.js

1 Service
0 Database

Processes and hosts

Process Production SampleDevOpsApp
Runs on Production
State Available

Automated Tag Rules: Define ONCE in Dynatrace!

#1

Settings > Tags > Automatically applied tags > DeploymentGroup

DeploymentGroup

#2 – Call the Tag “*DeploymentGroup*”

This custom tag is applied to all services, process groups, or hosts that match one or more of the rules defined below. Matching entities are listed at the bottom of this page.

It can take up to a minute for rules to detect newly added entities and changes to existing entities.

Add new rule

Rule

Services on process groups where *DEPLOYMENT_GROUP_NAME* (Environment) exists

Rule applies to Services of the following topology

Optional tag value `{ProcessGroup:Environment:DEPLOYMENT_GROUP_NAME}`

Available placeholders

- AwsRelationalDatabaseService:Endpoint*
- AwsRelationalDatabaseService:InstanceClass*
- AwsRelationalDatabaseService:Name*
- AzureRegion:Name*
- AzureScaleSet:Name*
- AzureVm:Name*
- CustomDevice:DetectedName*
- CustomDevice:DnsName*
- CustomDevice:IpAddress*
- CustomDevice:Port*
- DockerContainerGroupInstance:ContainerName*
- DockerContainerGroupInstance:FullImageName*
- DockerContainerGroupInstance:ImageVersion*

Move up/down Off/On Delete Edit

Rule applies to entities matching the following properties:

- All process groups
- All service types
- All service technologies
- All service topologies

Conditions:

All conditions listed below must be met for this rule to be applied.

DEPLOYMENT_GROUP_NAME (Environment) exists

Add condition

Apply tag to underlying process groups of matching services

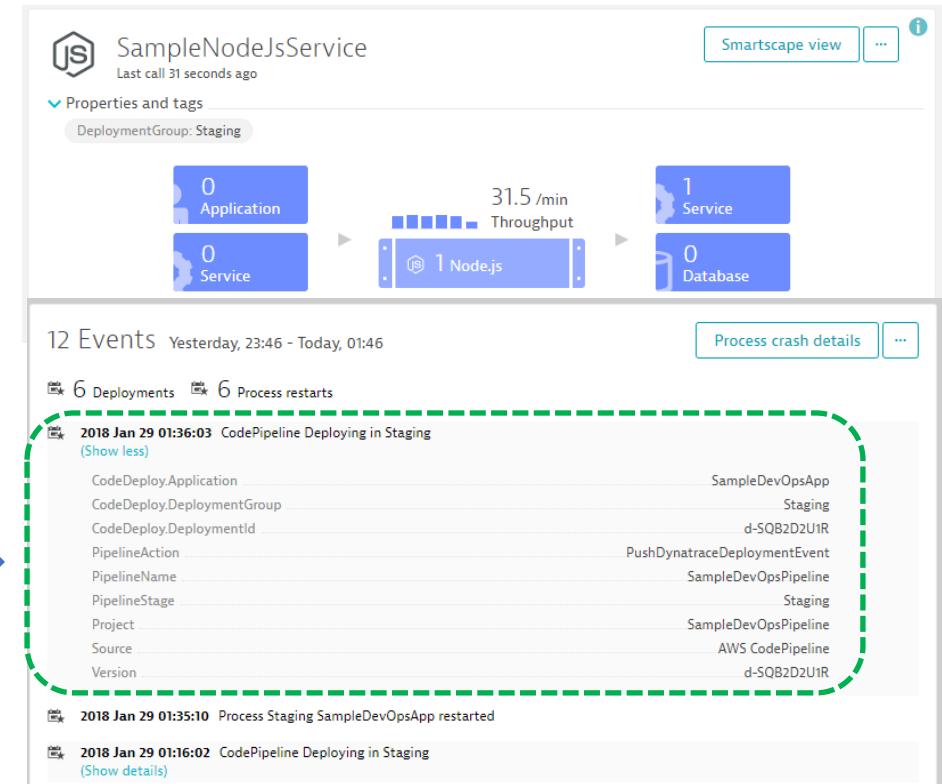
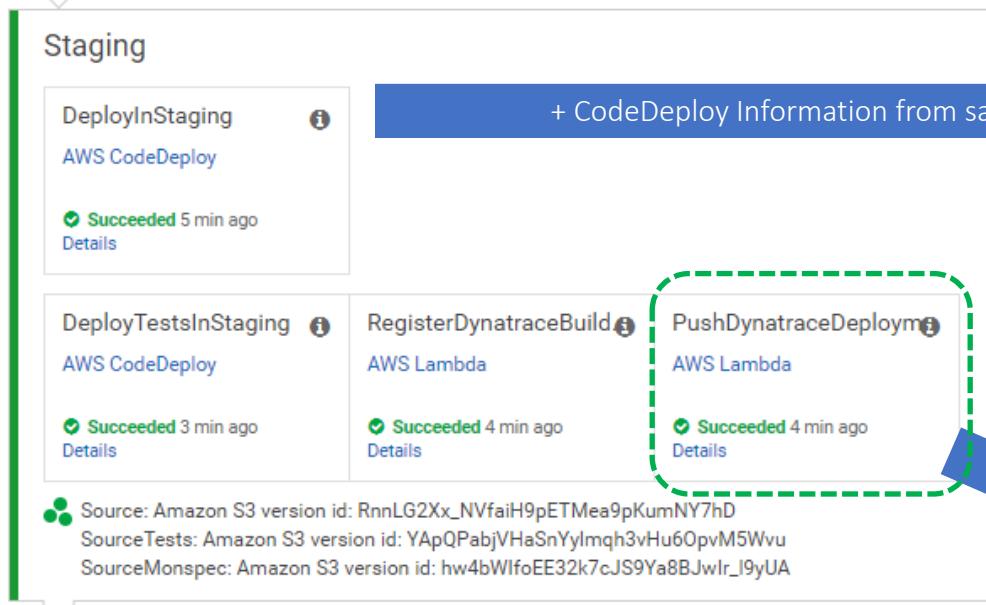
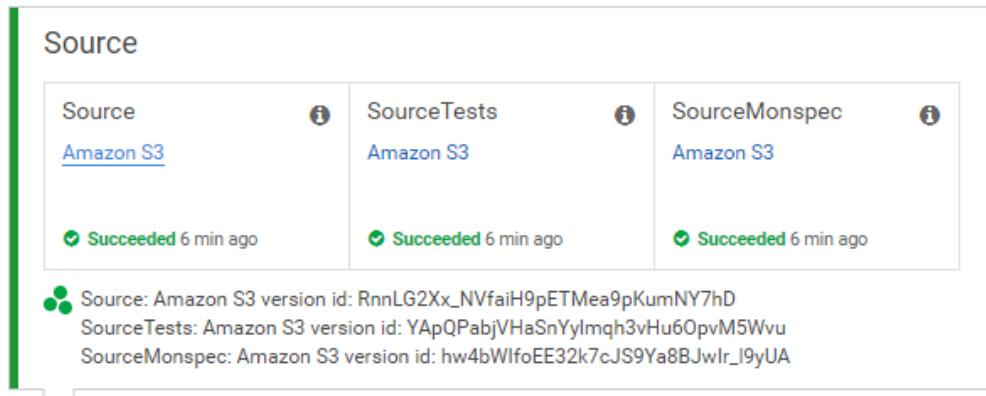
Apply tag to underlying hosts of matching services

#3 – For Services

#4 – Tag Value comes from our Process Environment Variable

#5 – Only apply rule if env variable exists

@Deployment: Pass Deployment Information to Deployed Entities



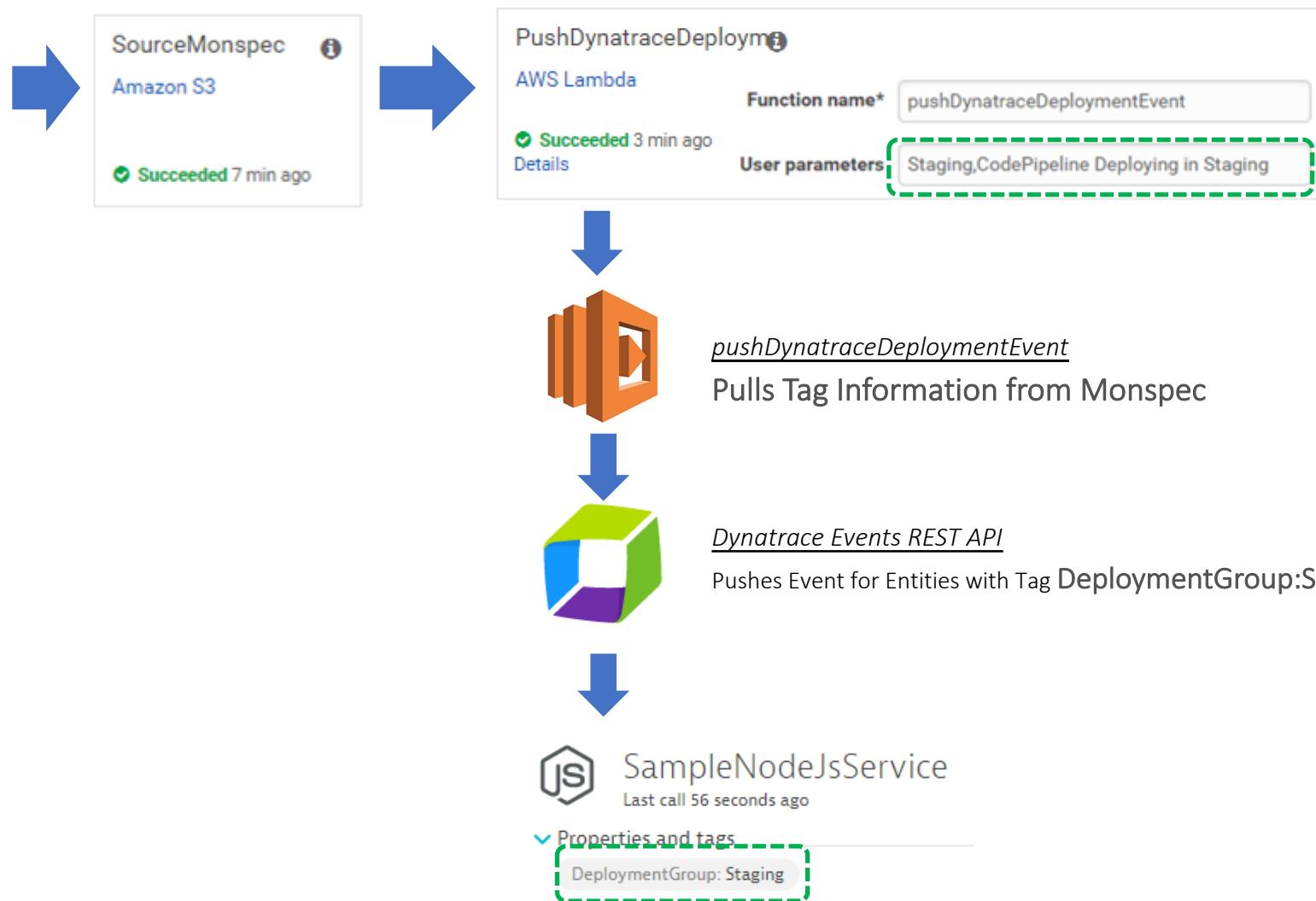
Dynatrace Events REST API



pushDynatraceDeploymentEvent
Pushes CodePipeline Info to Dynatrace

Monitoring as Code (Monspec): Define Services and their Environments as Code

```
Monspec (Monitoring as Code)
"SampleJsonService" : {
    "etype": "SERVICE",
    "environments" : {
        "Staging" : {
            "tags" : [
                {
                    "context": "CONTEXTLESS",
                    "key": "DeploymentGroup",
                    "value": "Staging"
                }
            ]
        },
        "Production" : {
            "tags" : [
                {
                    "context": "CONTEXTLESS",
                    "key": "DeploymentGroup",
                    "value": "Production"
                }
            ]
        }
    }
},
```



Behind the Scenes: Approval Stage

Shift-Left: Automate Metric Comparison to Validate Build

Monitoring as Code: Compare Metrics of Services of Different Environments/Builds

Entity: SampleJsonService

Environment: Staging
Tag: DeploymentGroup:Staging

Metrics: ResponseTime (Avg)

Compare: StagingToProduction
Source: Staging; Compare: Prod
Factor: Staging = **10% Slower**

Environment: Production
Tag: DeploymentGroup:Production

Metrics: ResponseTime(Max) < 10s

Compare: StagingToProdYesterday
Source: Staging; Compare: Prod
Factor: 10%, Timeshift: 86400s

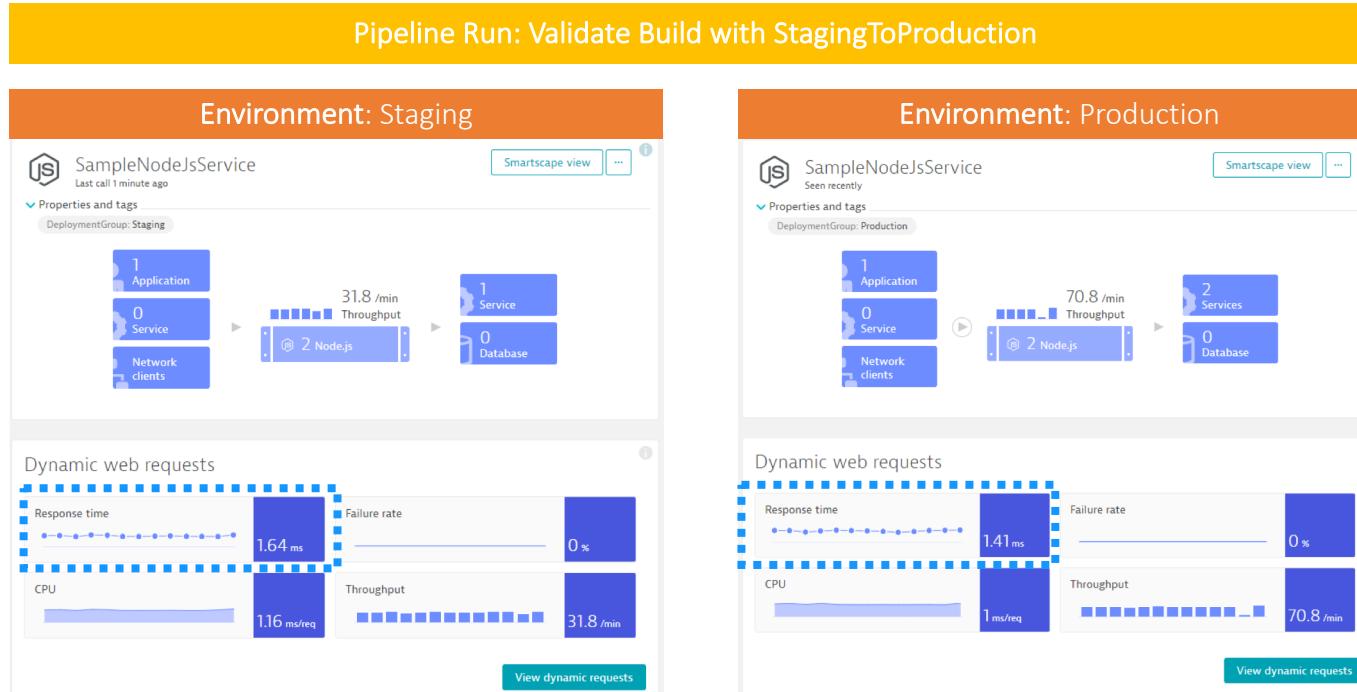
Metrics: FailureRate(Avg)

Compare: StagingToStagingLastHour
Source: Staging; Compare: Staging
Factor: 0; Timeshift: 3600s

Metrics: ResponseCount
Compare: StageToStageYesterday
Source: Staging; Compare: Staging
Factor: 0; Timeshift: 86400s

Different Metrics

Different Comparisons



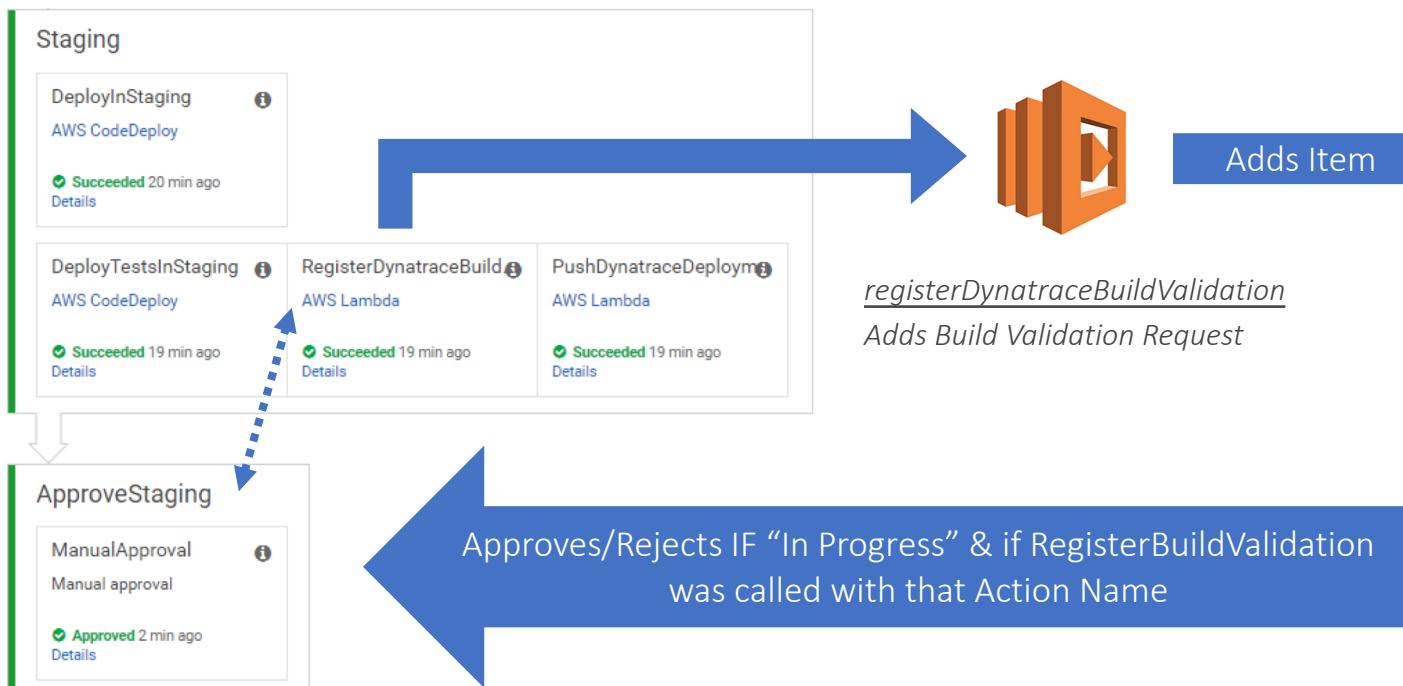
Source (Staging) : 1.64ms

Compare (Production) : 1.41ms

Threshold (+10%) : 1.56ms

Status: **VIOLATION**

@Approval: Automating Build Validation into your CodePipeline



registerDynatraceBuildValidation
Adds Build Validation Request



Monspec from S3



Build Validation Request Item

- Pipeline Information
- Monspec
- Timestamp + Timeframe
- Comparison Definition Name
- Action Name to Approve / Reject



Updates Build Validation Request

- Updated Monspec
- Updated Status



validateBuildDynatraceWork



CloudWatch Events
(e.g: 1min)



Dynatrace Entities & Timeseries REST API

Resolves Tags and gets list of Entities
Queries Metrics for these Entities

Monitoring as Code: The Full Picture

```
monspec.json
{
  "SampleJSonService" : {
    "etype": "SERVICE", // Options are SERVICE, APPLICATION
    "name": "SampleNodeJsService",
    "environments" : {
      "Staging" : {
        "tags" : [
          {
            "context": "CONTEXTLESS",
            "key": "DeploymentGroup",
            "value": "Staging"
          }
        ]
      },
      "Production" : {
        "tags" : [
          {
            "context": "CONTEXTLESS",
            "key": "DeploymentGroup",
            "value": "Production"
          }
        ]
      }
    }
  }
}
```

Entity: What Entity do we want to validate?

Environments: Tell us how we can detect these entities in Dynatrace

Monitoring as Code: The Full Picture

monspec.json

```
{  
  "SampleJsonService" : {  
    ...  
    "comparisons" : [  
      {  
        "name" : "StagingToProduction",  
        "source" : "Staging",  
        "compare" : "Production",  
        "scalefactorperc" : {  
          "default": 10,  
          "com.dynatrace.builtin:service.requestspermin" : 90  
        },  
        "shiftcomparetimeframe" : 0,  
        "shiftsourcetimeframe" : 0,  
      },  
      {  
        "name" : "StagingToProductionYesterday",  
        "source" : "Staging",  
        "compare" : "Production",  
        "scalefactorperc" : {  
          "default": 10,  
          "com.dynatrace.builtin:service.requestspermin" : 90  
        },  
        "shiftsourcetimeframe" : 0,  
        "shiftcomparetimeframe" : 86400  
      }  
    ]  
  }  
}
```

Comparision: Which environments do we compare and how!

monspec.json

```
{  
  "SampleJsonService" : {  
    ...  
    "comparisons" : [  
      {  
        "name" : "StagingToStagingLastHour",  
        "source" : "Staging",  
        "compare" : "Staging",  
        "scalefactorperc" : { "default": 0 },  
        "shiftsourcetimeframe" : 0,  
        "shiftcomparetimeframe" : 3600  
      },  
      {  
        "name" : "ProductionToProductionLastH",  
        "source" : "Production",  
        "compare" : "Production",  
        "scalefactorperc" : { "default": 0 },  
        "shiftsourcetimeframe" : 0,  
        "shiftcomparetimeframe" : 3600  
      }  
    ]  
  }  
}
```

Monitoring as Code: The Full Picture

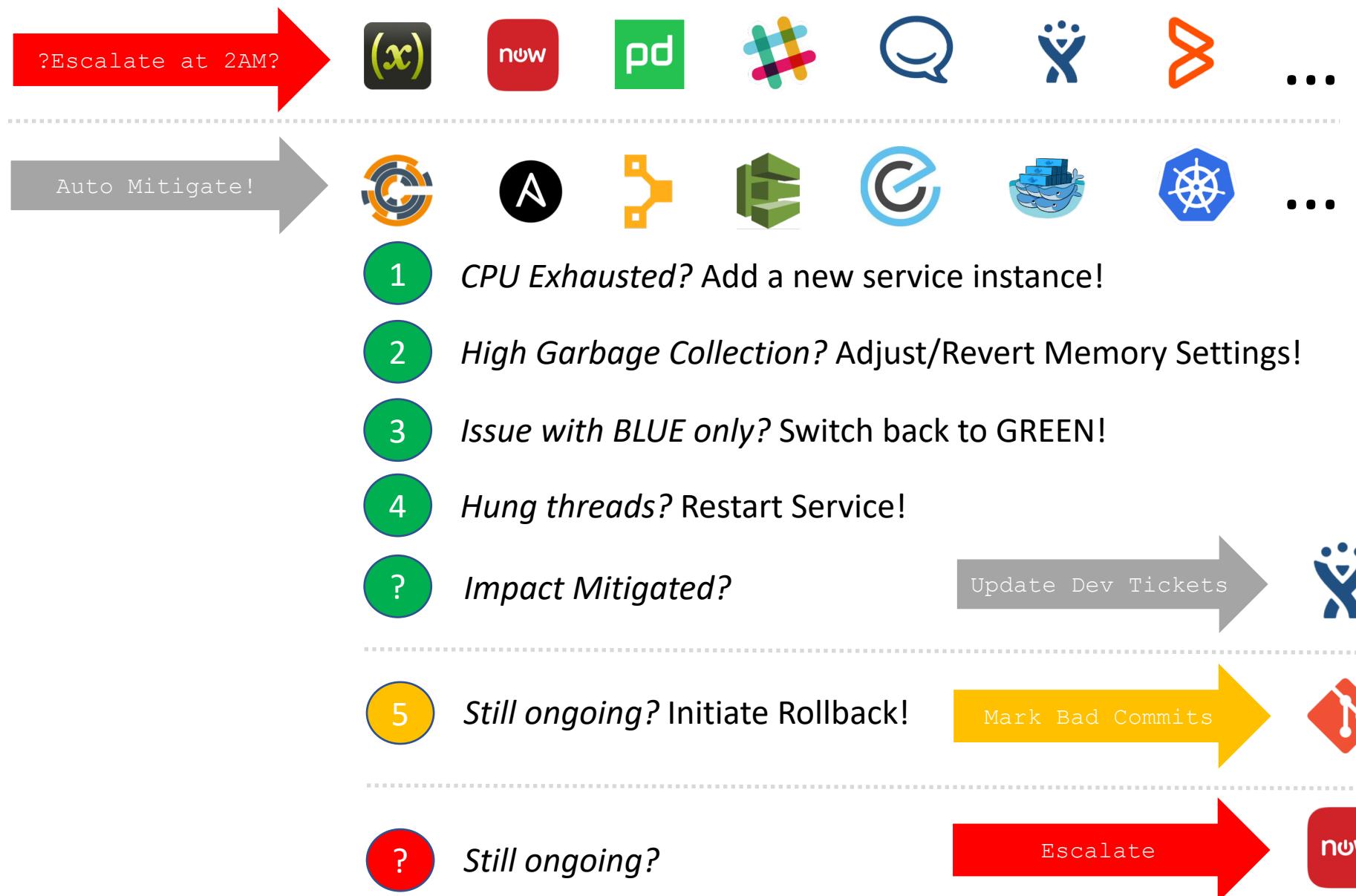
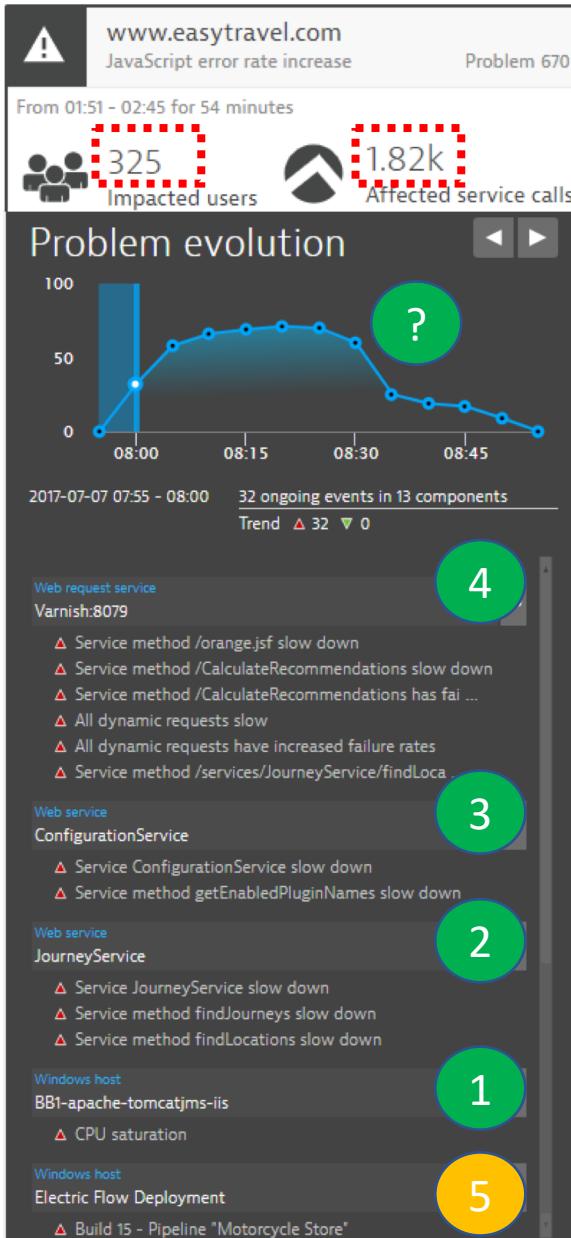
```
monspec.json
{
  "SampleJsonService" : {
    ...
    "perfsignature" : [
      {
        "timeseries" : "com.dynatrace.builtin:service.responsetime",
        "aggregate" : "avg", // min, max, avg, sum, median, count, percentile
        "validate" : "upper", // upper or lower
        // "upperlimit" : 100, // Optional: Can be used to define a FIXED THRESHOLD
        // "lowerlimit" : 50, // Optional: Can be used to define a FIXED THRESHOLD
      },
      {
        "timeseries" : "com.dynatrace.builtin:service.responsetime",
        "aggregate" : "p90"
      },
      {
        "timeseries" : "com.dynatrace.builtin:service.failurerate",
        "aggregate" : "avg"
      },
      {
        "timeseries" : "com.dynatrace.builtin:service.requestspermin",
        "aggregate" : "count",
        "validate" : "lower"
      }
    ],
  }
}
```

Metrics: Which Metrics, Aggregation, Upper/Lower Boundaries

Behind the Scenes: Self-Healing

Smart Remediation Action based on AI-Detected Problems

Path to NoOps: Better and Smarter Auto-Remediation, Self-Healing, ...



Automated Rollbacks with AWS Lambda and AWS CodeDeploy

Production

DeployInProduction AWS CodeDeploy	Succeeded 14 hours ago Details
DeployTestsInProduction AWS CodeDeploy	Succeeded 13 hours ago Details
PushDynatraceDeployment AWS Lambda	Succeeded 13 hours ago Details

#1: Push Deployment Information,
e.g: CodeDeploy DeploymentId

6 Events Yesterday, 16:24 - Today, 16:24

Process crash details ...

3 Process restarts 3 Deployments

2018 Jan 23 02:23:37 CodePipeline Deploying in Production (Show less)

CodeDeploy.Application	HelloWorld
CodeDeploy.DeploymentGroup	Staging
CodeDeploy.DeploymentId	d-YVV3NSFNP
PipelineAction	PushDynatraceDeploymentEvent
PipelineName	Demo1
PipelineStage	Production
Project	Demo1
Source	AWS CodePipeline
Version	d-YVV3NSFNP

Production/SampleNodeJsService Failure rate increase Problem 235

Production/SampleNodeJsService Failure rate increase Problem 235

From 02:02 - 02:41 for 39 minutes

Impacted services Production/SampleNodeJsService

Impact 93.2 requests/min affected

One comment

Triggering AWS AutoDeploy of previous DeploymentId: d-W0YYGEANP
agrabner@dynatrace.com, Jan 23, 2018 16:39

#2: Calling Lambda via API Gateway



handleDynatraceProblemNotification

#4: Redeploy previous revision



Uses Dynatrace Events API
to pull CUSTOM_DEPLOYMENT events

#5: Push comment to Dynatrace

Simulating Different “Bad” Builds

- Application looks at startup for an environment variable BUILD_NUMBER. Allowed values, 1, 2, 3, 4

Build Number	Behavior
1	No Problem
2	50% of requests return HTTP 500
3	No Problem
4	Staging OK. 10% Requests fail in Production

- To Deploy a different build:
 - Change the build number value in the associated Build Pipeline variable
 - Save and Queue
 - Deploy a new build (Release Pipeline)