HCL

DevOps

Case Study

Digital & Analytics
by HCL Technologies

Case Study:
Oil and Gas Company, UK

Client Background:

One of the world's largest integrated oil and gas companies, headquartered in UK

Area of Service:

Cloud Operations and DevOps on Azure

Nature of Engagement:

Long-term CloudOps Efficiency enhancement through DevOps Automation

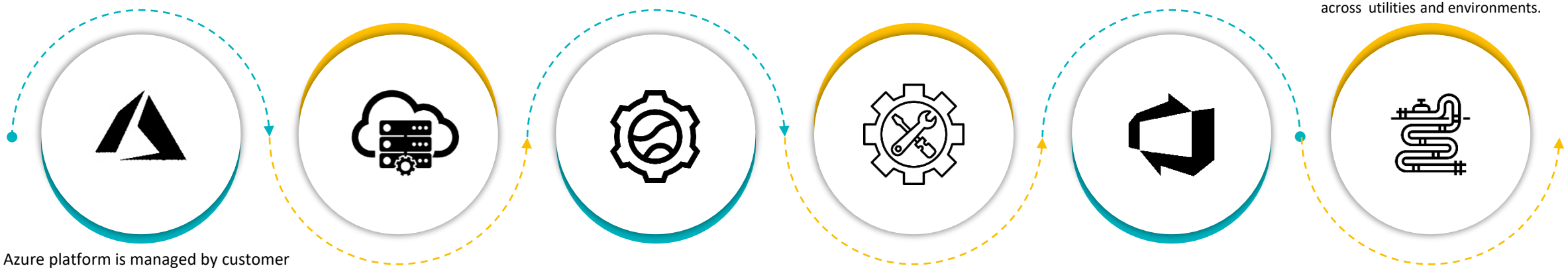
Background

IaaS / PaaS and Apps provisioning on cloud platform (mainly on Azure) for different customer divisions based on specific division's needs and privileges were required.

Number of utilities/components - Azure Runbooks, ARM Template, .NET API and Azure DevOps extensions were developed to automate the deployment of commonly used Azure resources

Classic Azure DevOps CI/CD pipelines to deploy each utility

- For each target environment there was a separate CI/CD pipeline per sub-component. As a result, there was a large number CI/CD pipelines (300+) across utilities and environments.



Azure platform is managed by customer with the help of multiple vendors - all Azure resource provisioning for different divisions / projects were done through Azure DevOps

Governance process (especially the deployment process) were not streamlined and monitored across multiple vendors

Azure DevOps extension was used to deploy Azure resources which in turn called runbook, .NET API and used ARM template to deploy Azure resources

Key Challenges:

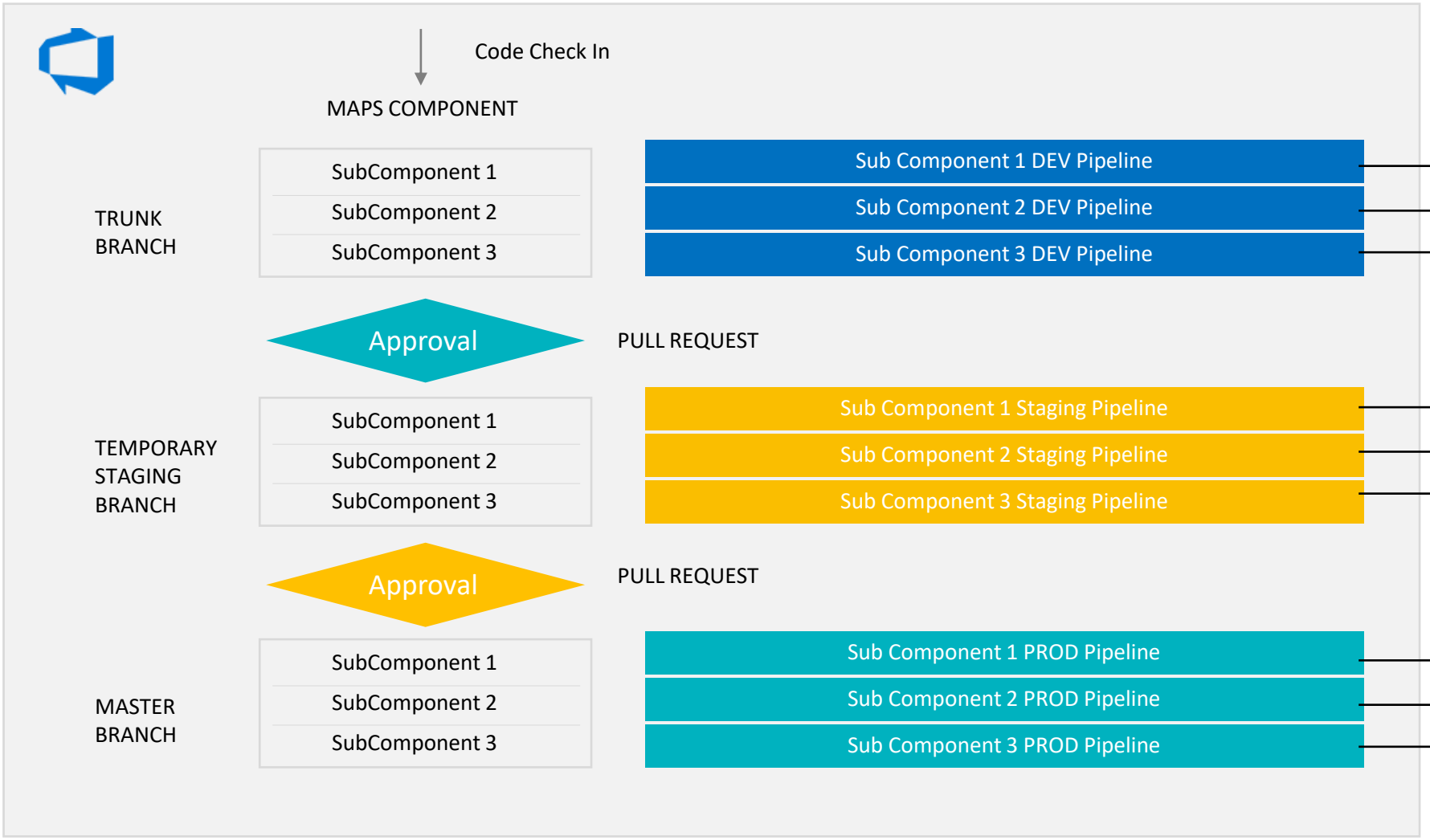
- Managing more than 300+ DevOps pipelines
- Unable to get a unified view of various DevOps projects and pipelines status
- Managing multiple variables in automation accounts
- Managing PS modules in automation accounts
- Keeping Wiki Portal up-to-date with frequently changing ARM templates
- Tracking of changes - Who, What, When, Where, and Why

Our Solution

We provided a holistic approach to firm up the Deployment process:
Analyzed existing pipelines
Designed a Consolidation approach to consolidate multiple pipelines (refer next slide)
Created a prototype of new approach to prove the approach works
Extended existing DevOps pipelines to include quality check gates, approval process specific to each division
Proposed a new Branching strategy (refer to Branching strategy slide) to facilitate by-directional commits and pulls, create features from trunk, gated release to Production, traceability of backlogs and bug fix commits
Identified critical manual tasks across DevOps processes and provided automation solution for each of those tasks - reducing errors, reducing maintenance and to speed up the deployment process
Assessed customer's current DevOps maturity through our in-house EAZe Framework
Assessed all pipelines and provided recommendation on Key DevOps Metrics to be captured across environments / pipelines to have better DevOps Governance
Implemented custom DevOps Dashboards showcasing all the metrics that we recommended across pipelines and environment

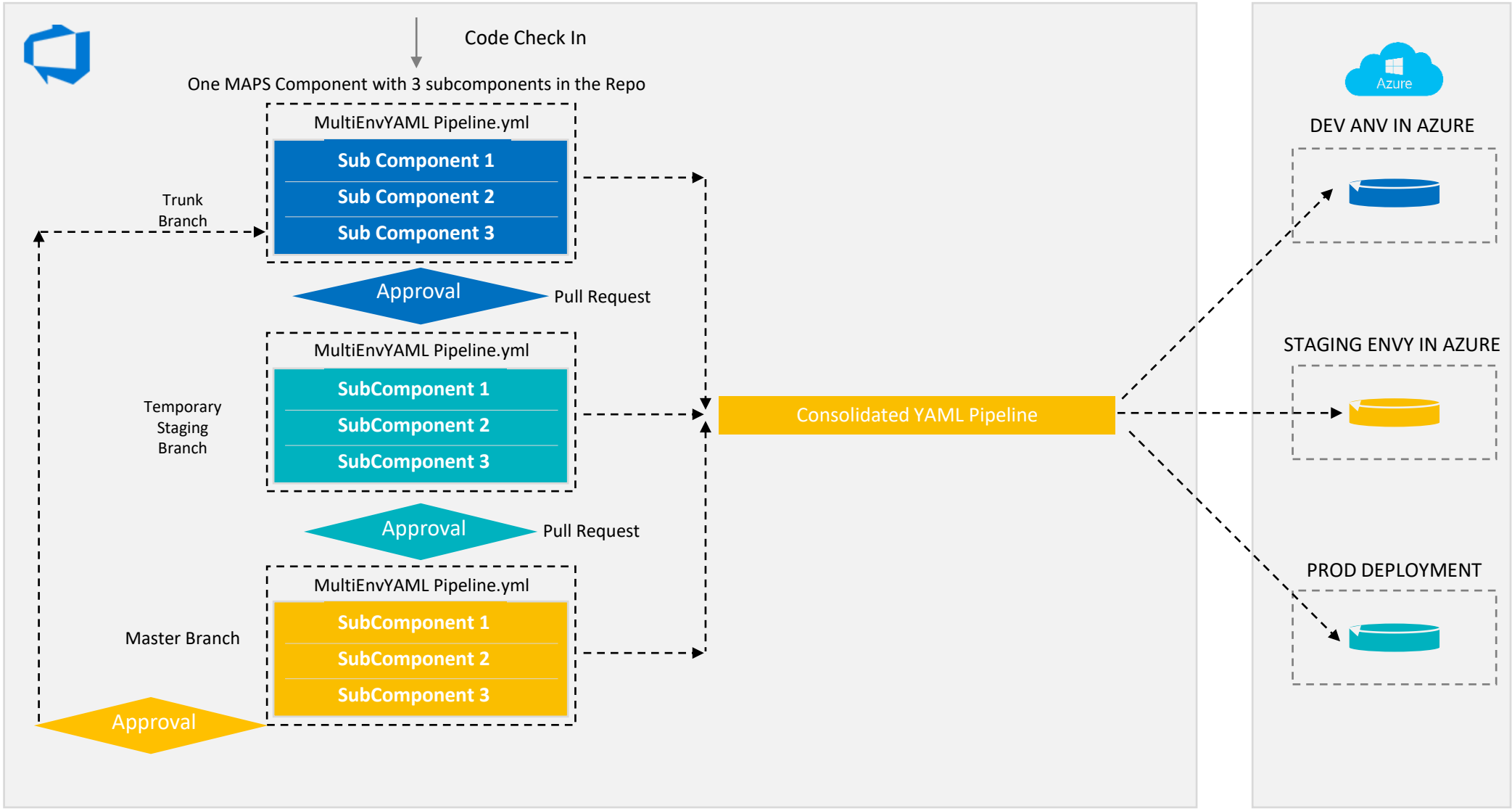
Pipeline Consolidation and Branching Strategy

Before Pipeline Consolidation



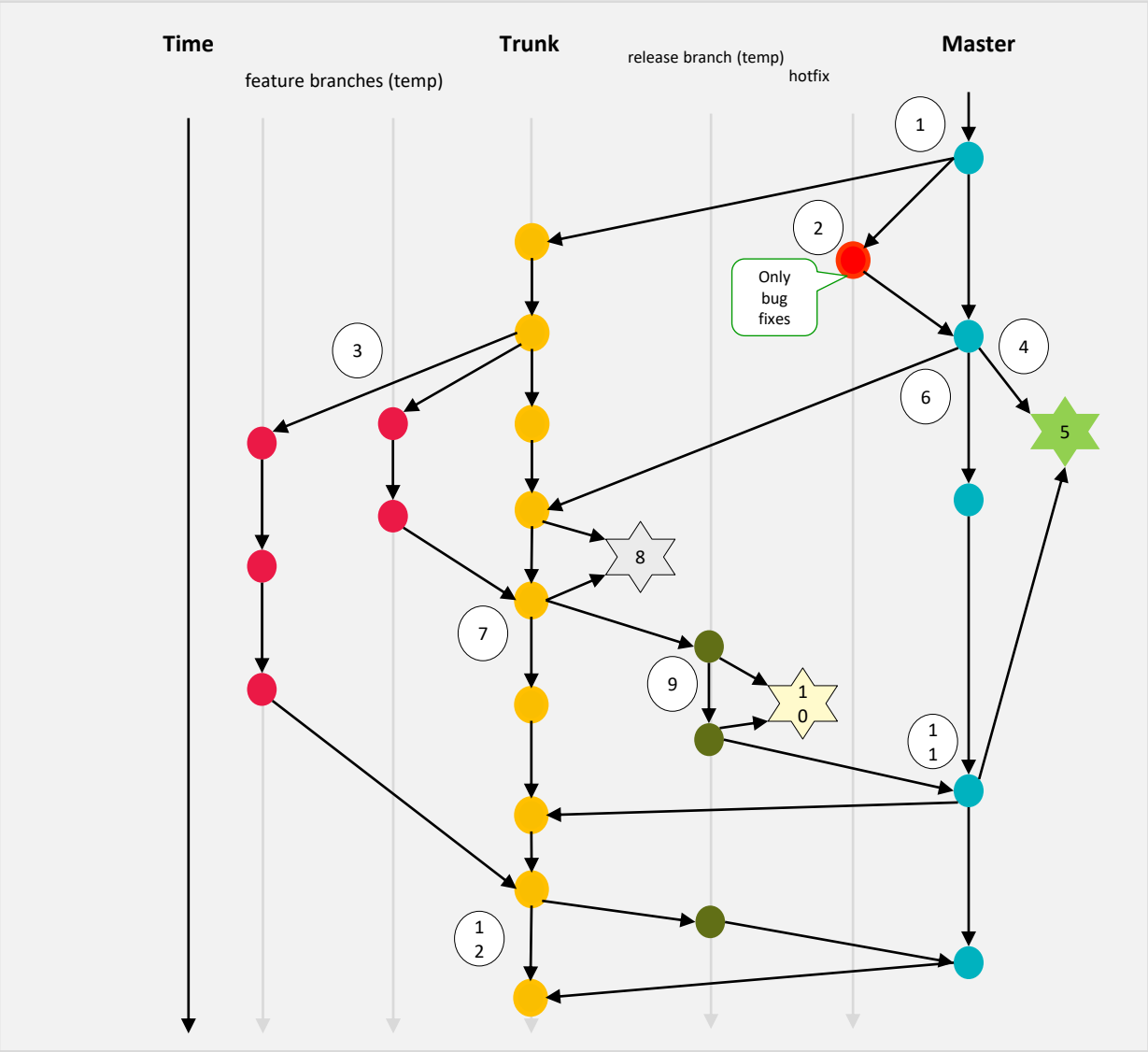
- Each main component had multiple sub-components
- For each sub-components and per Environment one Pipeline was created
- No Bidirectional updates - Changes done in Master branch for Production not reflecting in Trunk/Staging branches
- Pull to Trunk or Pull to Dev not available
- No feature branches possible from Trunk
- No traceability of bug fix backlogs to commits

Pipeline Consolidation Approach



- Our approach was to implement YAML pipelines
- Single YAML pipeline per package with all subcomponents
- Same pipeline to contain deployment scripts for all the three target environments (Dev, Staging, Prod)
- Dev environment to maintain versions in Trunk Branch
- Staging to maintain in Staging Branch
- Production Environment to maintain versions in Master Branch

Branching Strategy Approach



Approach

- Master is all synchronized with trunk
- Temporary hotfix branches are allowed for urgent fixes only
- Developers can create feature branches from trunk
- Gated pull to Master
- Deployment to Production
- Master is automatically pulled to Trunk after Production release
- Developer responsibility to resync and pull to dev
- Gated pulls to Trunk and deployment to Development environment
- Gated approval for Release branch creation; Trunk is locked automatically
- Bug fixes can be deployed to staging from release branch
- Gated pull to Master branch
- CD to Production
- Pull to Trunk
- Automated deletion of temporary release branches
- Other enhancement will be pulled into dev only after code synchronization

Deployment through new Approach

- Runbooks were deployed to Automation account
- .NET API were deployed Azure App Service
- Policy definitions were deployed and assigned to Management group/subscription/resource group scope
- Private build agents are used for pipeline deployment
- Testing involved using Azure DevOps extension for the utility to create/provision particular Azure resource(Key Vault, Azure SQL, App Service Plan, App Service Environment etc.).
- Raising service request in Service Now which would then trigger the workflow for resource creation.

- ARM templates were deployed to storage accounts
- Azure DevOps Extension were deployed to Visual Studio Market place
- For staging and production, each subcomponent was deployed in five regions like North Europe, North Central US, West Europe, South Central US and South-East Asia.
- Deployment into staging and production triggered through approval process
- Resource Creation Options:
- In some cases, resource gets provisioned through a triggering of approval request emails and approval from approvers

Execution Approach and Timelines

Analyze existing pipelines / packages / Environments

Design consolidation strategy

Execute – Consolidate DevOps pipelines

Select

Shortlist components for consolidation based on below factors:

- Status of backlog
- Current deployment stage
- Complexity
- Dependency
- Business Priority
- Readiness

Create

Create new pipeline through YAML script

Consider all variation of tasks, quality check gates, variable groups, testing tasks across environments and accommodate them in YAML

Configure

Configure / realign existing packages with new branching strategy

Configure pipeline pull requests for each environment appropriately

Test

Test Environment / PaaS / IaaS service provisioning through new pipeline
Trigger each pull request from respective branch for environments
Validate user/role specific provisioning
SAST / DAST Testing

Bug Fix

Fix issues in new consolidated pipeline

Fix Environment specific issues

Fix issues with service connection name

Release

Archive existing classic Pipeline

Release the new Consolidated pipeline for developer's usage

Monitor the pipeline

Sprint based execution - Span of 2-4 weeks per component – varies based on complexity of pipeline

Automate DevOps Manual tasks

Automation Activities

There were multiple manual tasks carried out as part of DevOps process earlier.
We assessed all the manual tasks and identified three critical manual activities and automated such activities for better maintenance and to accommodate changes in a faster way.

1 ACTIVITY

- Azure Automation Account - Update All PowerShell Modules to latest version
- Azure Automation Account - Update Single PowerShell Module to a specific version

2 ACTIVITY

Update ARM templates in Wiki(Archetypes)

3 ACTIVITY

Update variables in Automation account

Automation Activities – scope and approach

Activity	Scope	Approach Followed
Activity 1:PowerShell Module Update	❖ Update PowerShell Module in automation account from outside the automation account ❖ Governance policy should be excluded and included before and after updation of PowerShell module ❖ There should be provision to update all modules to latest version and also to update single module to particular version	❖ Import scripts for PowerShell module update as run-book in the Automation Account ❖ Create Web Hook for the imported run-books ❖ Create YAML pipelines for all module update and single module update ❖ Tasks in the pipeline ○ Exclude Governance Policy ○ Call Web-hook for PowerShell module update ○ Include Governance Policy
Activity 2 : ARM template update	When ARM template gets deployed through CI CD pipeline in prod then template should be updated in Wiki	❖ Execute git bash commands from YAML pipeline ❖ Using git bash commands where there will be two remote repos (one source repo and another target wiki repo). ARM template will be copied from source to target repo
Activity 3 : Variable update	Exclude and Include governance policy in automation account Provision to add/update/delete variables in automation account	❖ Run PowerShell commands from pipeline to add/update/delete variables ❖ Exclude and Include governance policy in automation account

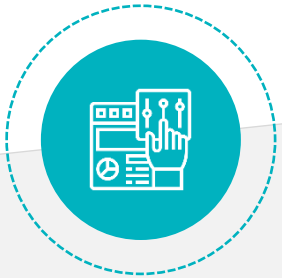
DevOps Metrics and Dashboard

DevOps Metrics Recommended

❖ No common DevOps metrics were measured or monitored across Azure DevOps pipelines.
❖ We assessed different sets of pipelines and recommended below metrics to be captured across components/ packages, pipelines and environments.

Pipeline Run Specific Metrics	Build, Release And Deployment
❖ Pipeline Pass Rate	❖ Active releases
❖ Pipeline Pass Rate Trend	❖ Deployment Success/Failure (% Failure can be calculated if we have sufficient data)
❖ Pipeline Failure Trend	❖ Builds by result.
❖ Pipeline Duration 80th Percentile	❖ Builds by Repository and branches
❖ Top 10 steps by Duration 80th Percentile	
Commits And Pull Requests	**Product Backlog Specific Metrics**
❖ Commits by Repository and branch	❖ Product Backlog by Type (Features/CR/Bugs/Tasks)
❖ Pull Requests by Repository for the commits	❖ Bug trend
❖ Related Work Items to the Pull Requests and Commits	❖ Cumulative Flow Diagram (CFD)
❖ Deployment frequency	❖ Lead/Cycle Time
❖ Deployment duration	
❖ Lead time – from Dev to deploy	
❖ Mean Time to recovery	

DevOps Dashboard



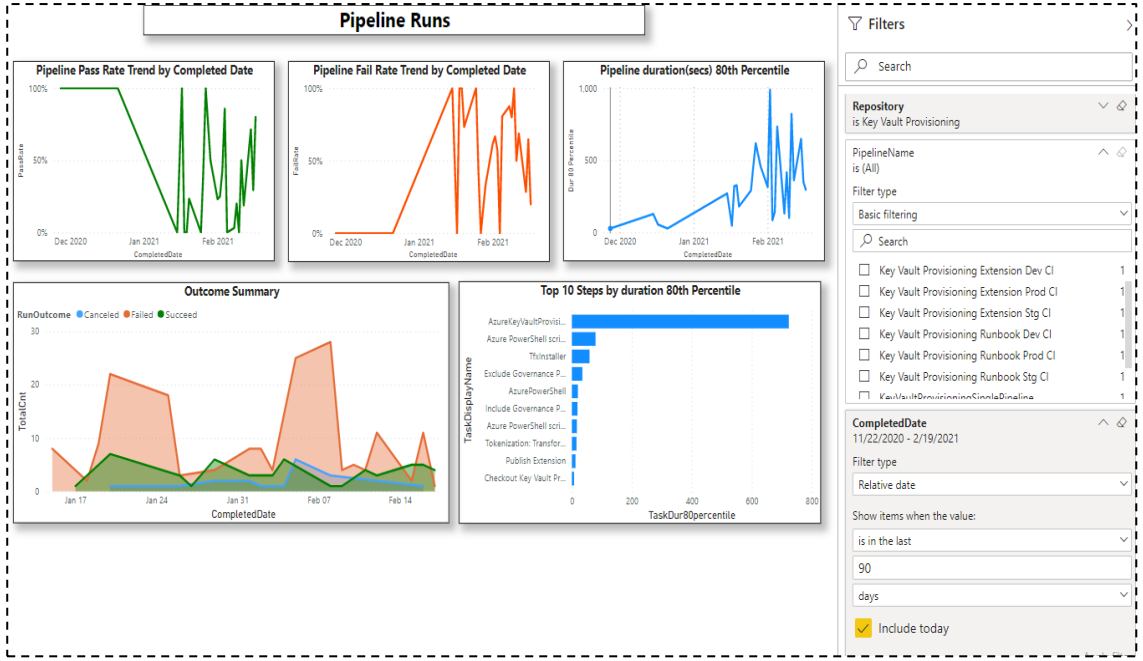
Customer required very interactive dashboards to filter/collapse/roll-up/roll-down metrics data at branch level / repository level/ package level

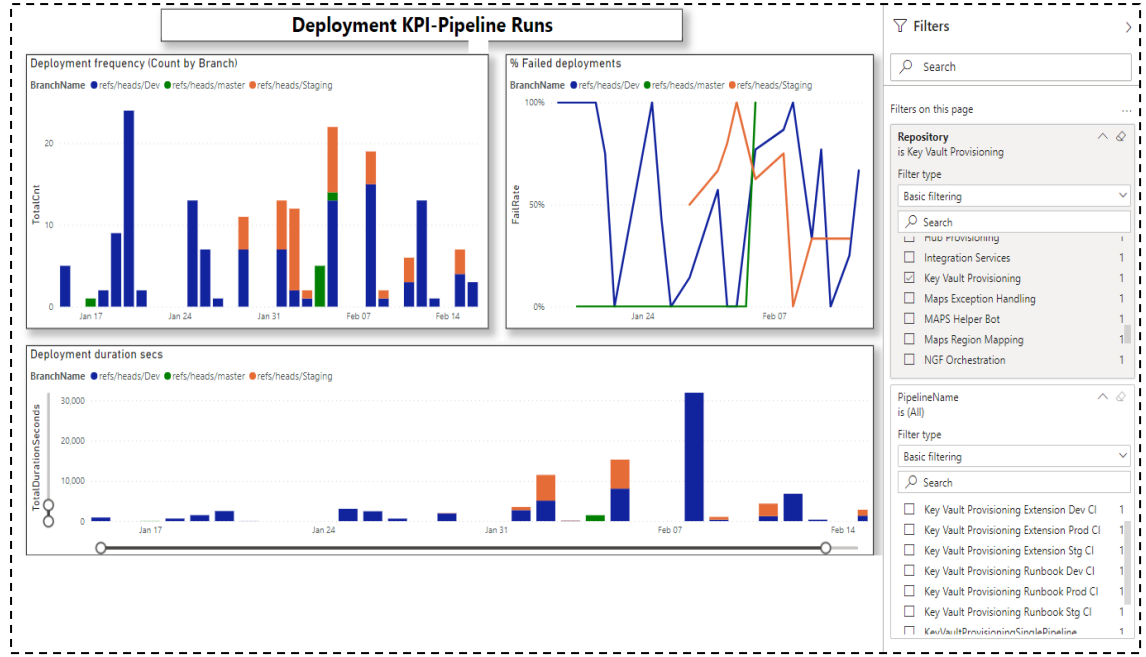
We leveraged Azure DevOps APIs to pull-in data relevant to all deployments across components/environments/commits

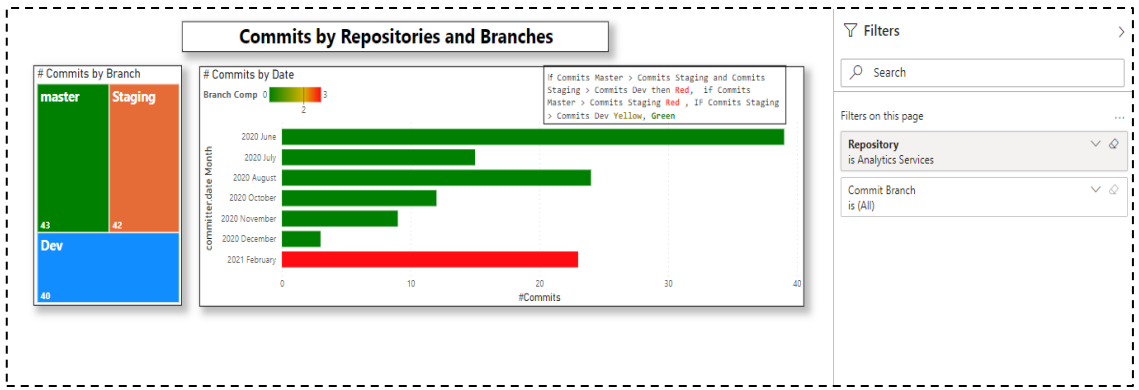
Created custom Power BI based widgets by creating multiple queries cross querying across various entities

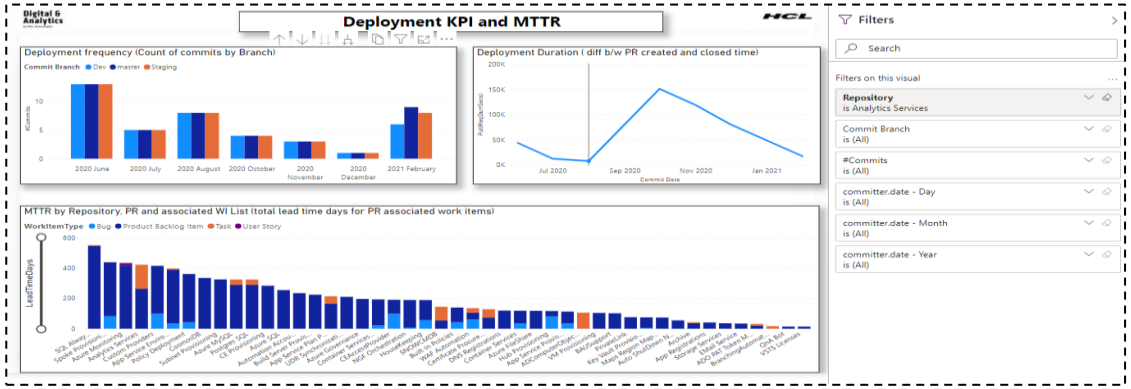
We were able to showcase many different custom views which were not possible through OOTB Azure DevOps dashboards

PBI Dashboard - View 1

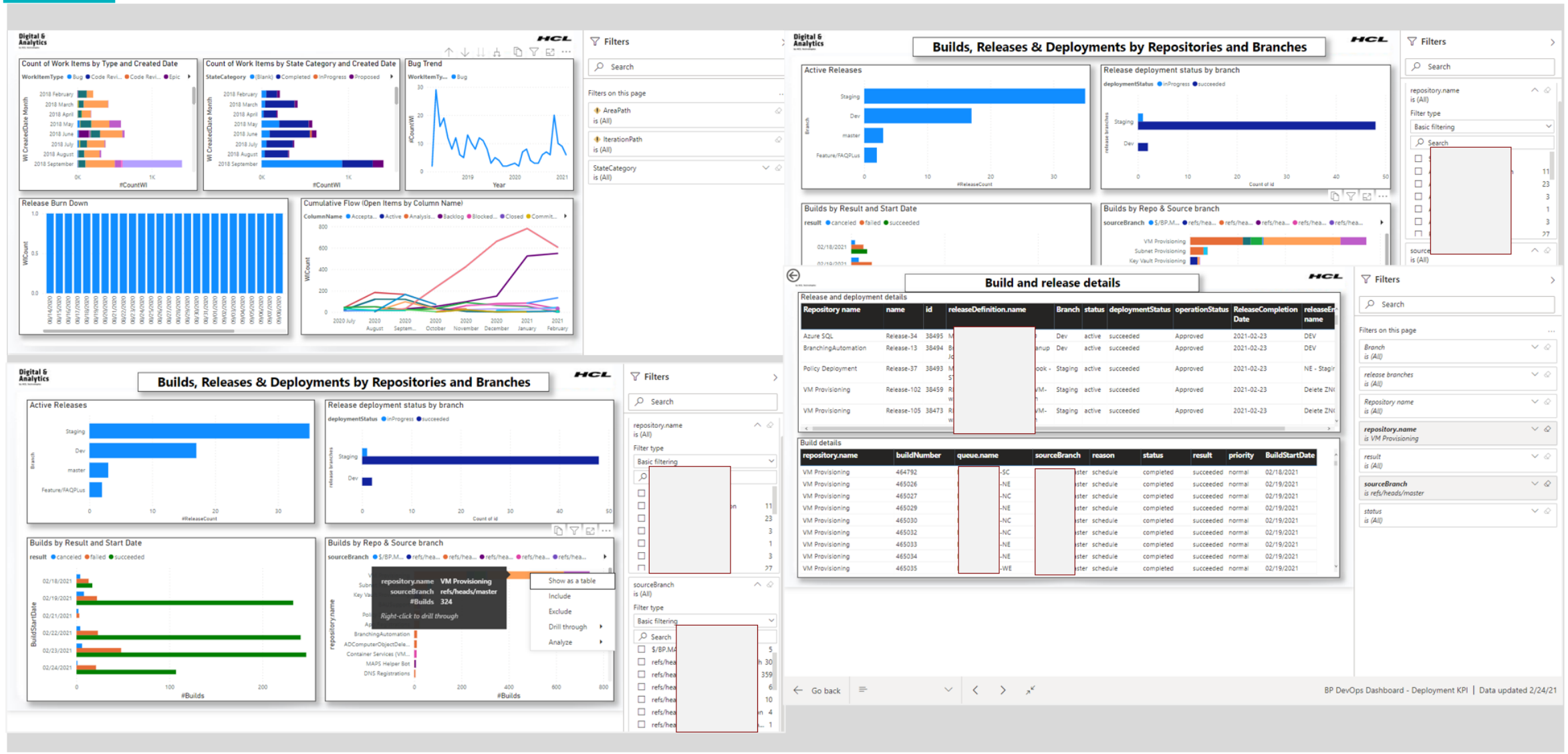






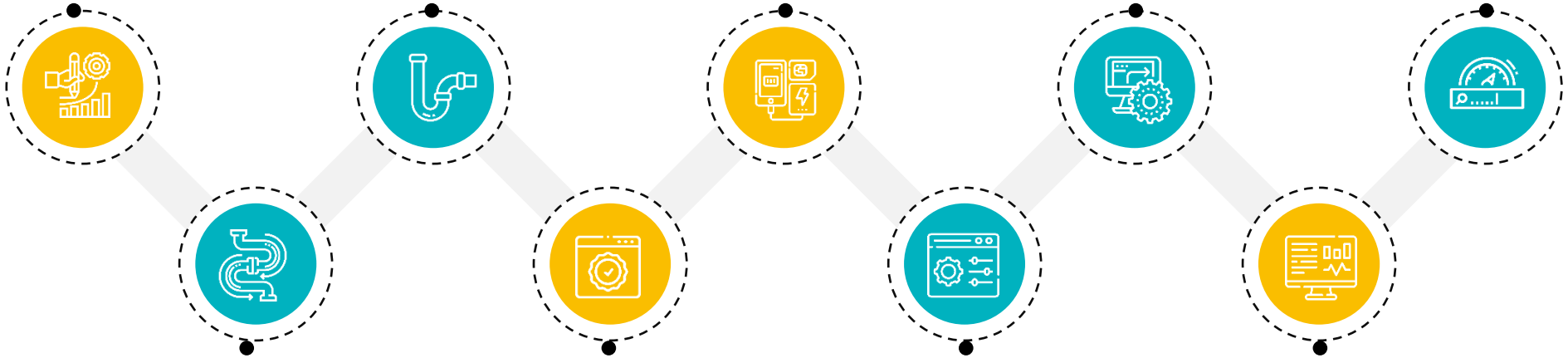


PBI Dashboard - View 2



Benefits

Benefits to the Customer



HCL

Relationship™

BEYOND THE CONTRACT

$10.5 BILLION | 176,000+ IDEAPRENEURS | 50 COUNTRIES