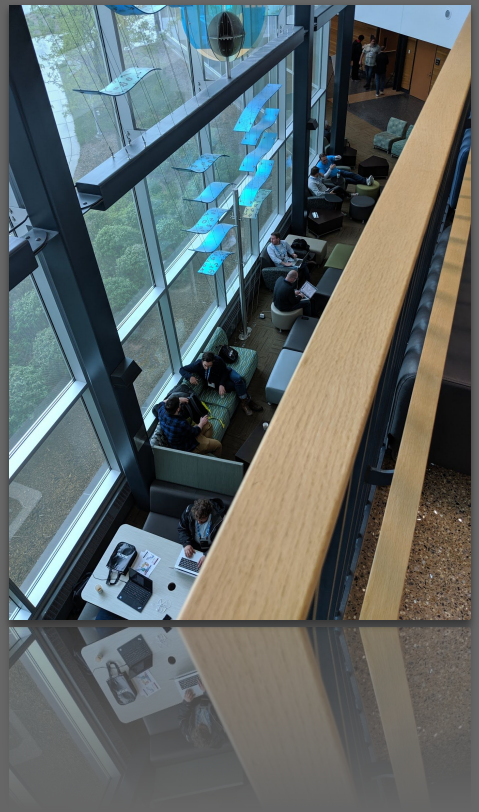
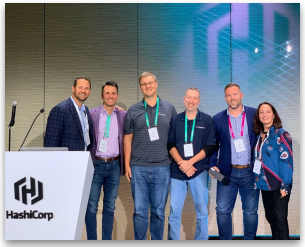
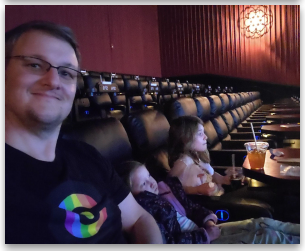


Full Stack CICD of Kubernetes Microservices using DevOps and IaC

Isaac Johnson @Medtronic





About Me

- Originally from Minnesota
  - San Diego 2010-2017.
- Family guy who really misses movies at the Alamo Drafthouse
- Enjoys camping, fishing and being outdoors
- Death Wish Coffee and insanely spicy foods
- In free time, creates courses for WhizLabs on DevOps
- Started Career at Control Data
- Principal SW Engineer at Medtronic
  - *however, I'm Cloud Solutions Architect at heart*

Disclaimer

The views and opinions expressed in this Open Source North talk are those of the authors (Isaac Johnson) and do not necessarily reflect the official policy or position of his employer, family, friends, denomination, country, place of origin, government, or pretty much anyone else.

Any content provided by Isaac Johnson are of his opinion and are not intended to malign any religion, ethnic group, club, organization, company, individual or anyone or anything. These opinions are not that of Medtronic. In fact, chances are they are well intentioned but likely flawed in some fashion by years of high test coffee consumption and EDM piped through headphones at ridiculous decibels.

Problem Statement

Requirements

We need to create a full containerized microservice platform using **Kubernetes**.

We need to use **Azure DevOps** but prefer to leverage **Open Source** solutions.

We need to **document** all that we do (so that we may scale)

Desires

We want corn fields, not flower boxes (for my Vegan friends)

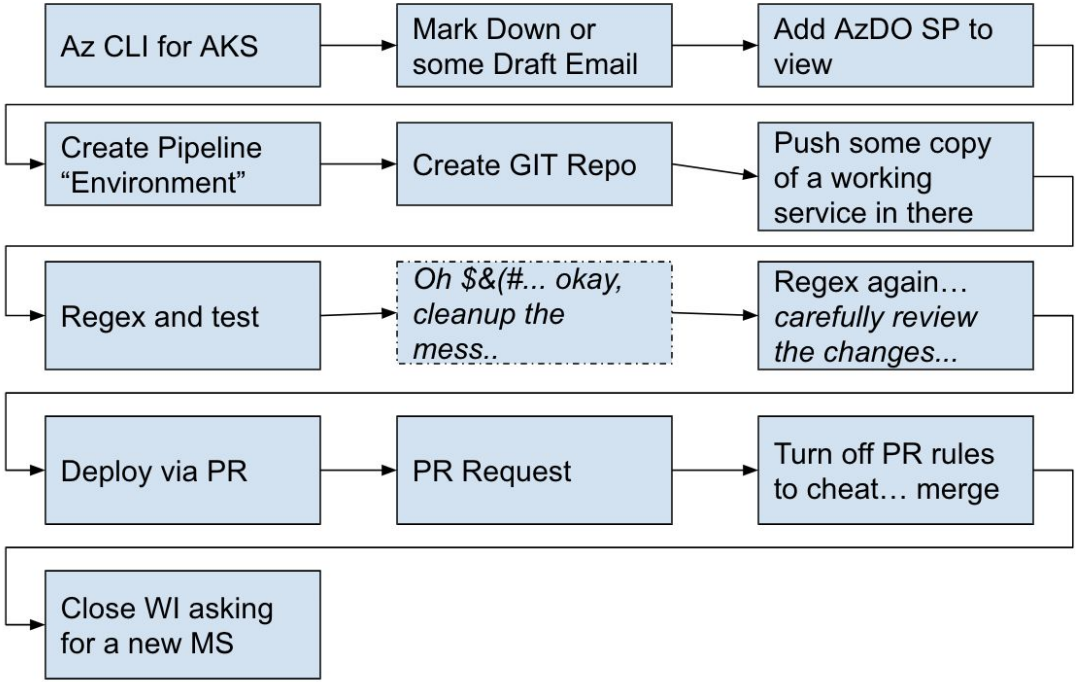
We will make mistakes: We want to fail fast and alert quickly.

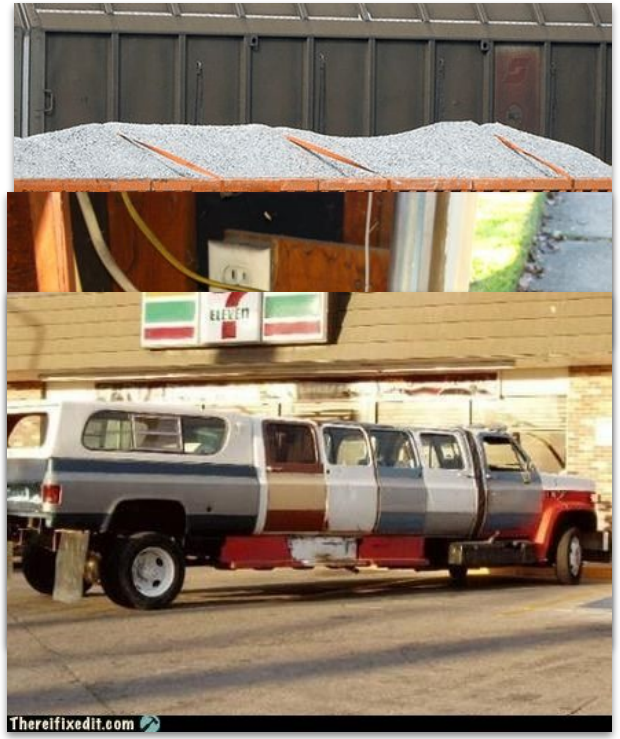
We want to leverage PaaS and SaaS.

Proposed Solution

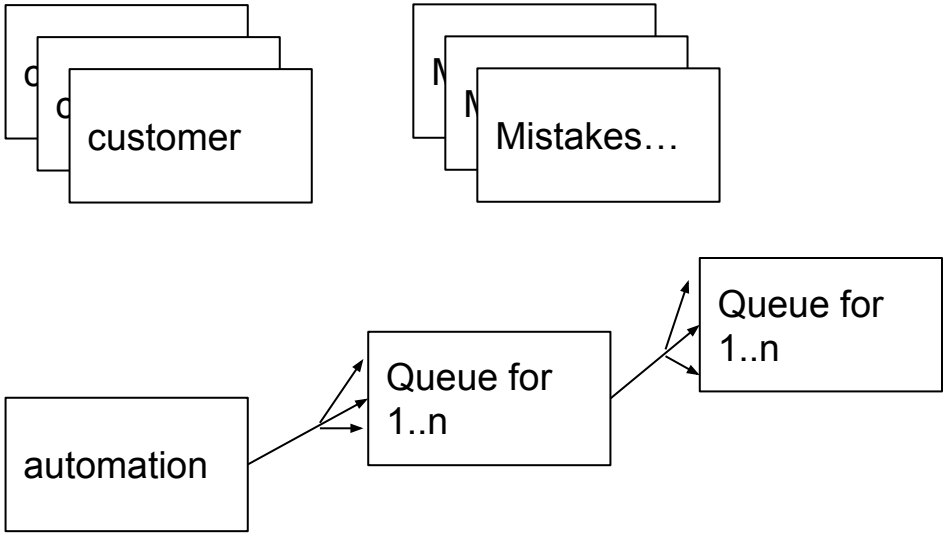
- Azure DevOps YAML Pipelines.
  - We should treat our build layer with the same review and policy rigor as our running code.
- Azure VMSS Agent pools and Azure Pipeline Pools
  - No special build agents that have been hand altered
- Azure Key Vault for Secrets
  - Leverage via YAML tasks and “Library” for classic pipelines
- Azure Kubernetes Service with AAD RBAC
  - Integrate with existing ID Provider
- ACR for Container and Helm storage
- Hashi Terraform for IaC layer

In the beginning...



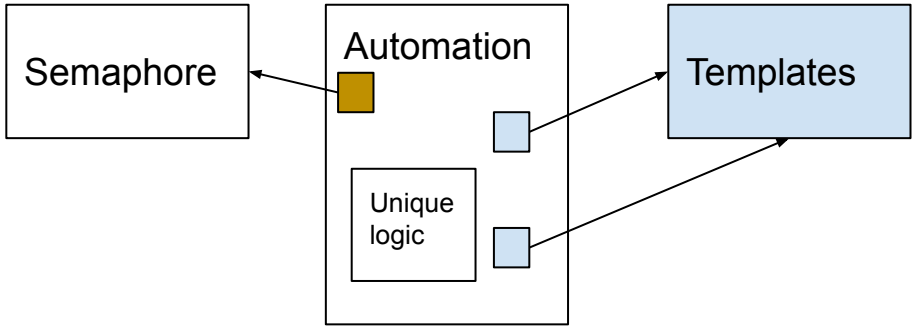


When Automations Fall Down...



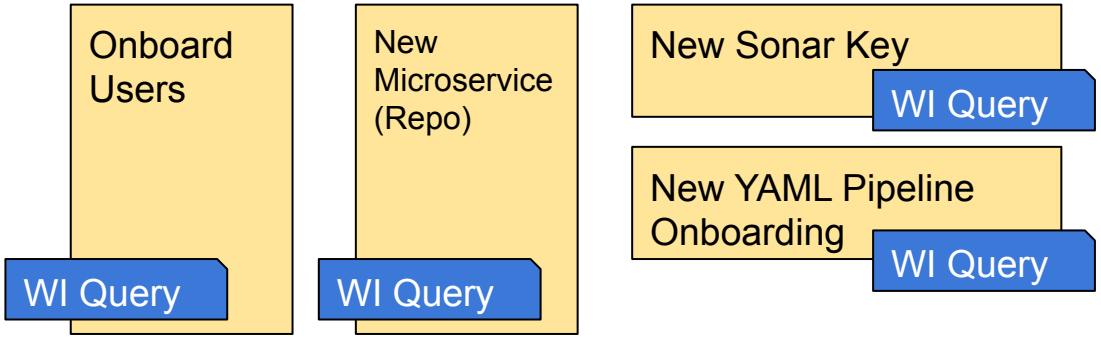
Pipelines running:	1	10	20
	10	100	2000

Rethinking things



Semaphore: variable or abstract data type used to control access to a common resource by multiple processes and avoid critical section problems in a concurrent system such as a multitasking operating system

Some WI Driven Automations



Templates: define reusable content, logic, and parameters. Templates function in two ways. You can insert reusable content with a template or you can use a template to control what is allowed in a pipeline

Semaphore

https://dev.azure.com/OURORG/OurProject/_apps/hub/ms.vss-build-web.ci-designer-hub?pipelineId=1234&branch=wia-processusers

```
- task: AzureCLI@2
          displayName: 'Azure CLI - wiq OnboardingsToProcess'
          inputs:
            azureSubscription: 'DevOps Environment (f492bf32-ca57-4f93-97a
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: 'az boards query --organization https://dev.azure.com/OURORG/ --id $(myWIQuerID) -o json  | jq ''.[] |
.id'' | tr ''\n'' '','' > ids.txt'
          env:
            AZURE_DEVOPS_EXT_PAT: $(DevOpsPAT)
- task: AzureCLI@2
          displayName: 'Azure CLI - Pipeline Semaphore'
          inputs:
            azureSubscription: 'Azure Environment (asdf-asdf-asdf-asdf-asdf)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: 'az pipelines build list --project OurProject --definition-ids 1234 --org https://dev.azure.com/OURORG/ -o
table > $(Build.StagingDirectory)/pipelinestate.txt'
          env:
            AZURE_DEVOPS_EXT_PAT: $(DevOpsPAT)
```

A WI Query into just a string of CSV

Check the states of running instances of *this* pipeline

Setting up CSV/IDs

```
- bash: |
    #!/bin/bash
    set +x
    # take comma sep list and set a var (remove trailing comma if there)
    echo "##vso[task.setvariable variable=WISTOPROCESS]"`cat ids.txt | sed 's/,$//'` > t.o
    set -x
    cat t.o
  displayName: 'Set WISTOPROCESS'
- bash: |
   set +x

   export IFS=","
   read -a strarr <<< "$(WISTOPROCESS)"

   # Print each value of the array by using the loop
   export tval="{"

   for val in "${strarr[@]}";
   do
     export tval="${tval}'process$val':{'wi':'$val'}, "
   done
```

Take IDs from our WI Query as a CSV and make an AzDO Var

Turn a CSV string into JSON block: *processXX{wi:XX}*

Using Semaphore

Null JSON on empty set

```
if [[ "$(WISTOPROCESS)" == "" ]]; then
    echo "##vso[task.setvariable variable=mywis;isOutput=true]{}" > ./t.o
else
   echo "##vso[task.setvariable variable=mywis;isOutput=true]$tval" | sed 's/..$/}/' > ./t.o
fi

# regardless of above, if we detect another queued "notStarted" or "inProgress" job, just die.. don't double process
# this way if an existing job is taking a while, we just bail out on subsequent builds (gracefully)
export tVarNS="`cat $(Build.StagingDirectory)/pipelinestate.txt | grep -v $(Build.BuildID) | grep notStarted | head -n1 | tr -d
'\n'`"
export tVarIP="`cat $(Build.StagingDirectory)/pipelinestate.txt | grep -v $(Build.BuildID) | grep inProgress | head -n1 | tr -d
'\n'`"

if [[ "$tVarNS" == "" ]]; then
    echo "No one else is NotStarted"
else
    echo "##vso[task.setvariable variable=mywis;isOutput=true]{}" > ./t.o
fi
if [[ "$tVarIP" == "" ]]; then
    echo "No one else is InProgress"
else
  echo "##vso[task.setvariable variable=mywis;isOutput=true]{}" > ./t.o
fi

set -x
cat ./t.o
```

Any other running or queued pipeline, then empty this one (make it a no-op)

Matrix on JSON (0..n)

```
    - job: runner
      dependsOn: parse_work_item
      strategy:
        matrix: $[
dependencies.parse_work_item.outputs['mtrx.mywis']]
```

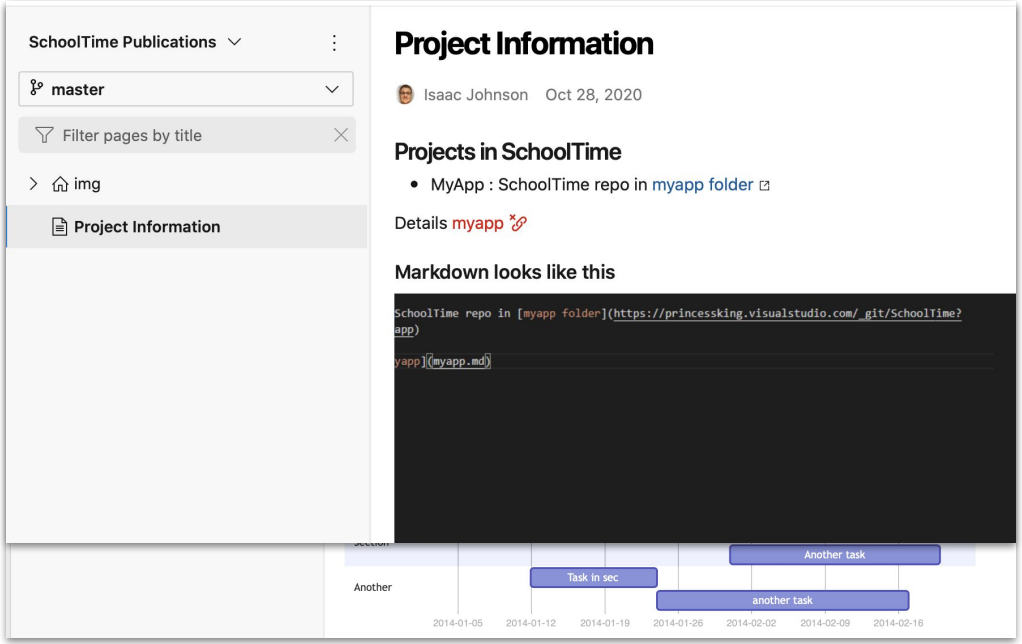
A typical IaC Repo

OurProjectAKS

- configure/
  - istio/
  - 010-cluster-roles.yaml
  - 020-companyspecific.yaml
  - 030-azdoagents.yaml
  - 040-aad-groups.yaml
  - 050-nginx-ingress.yaml
- docs/
  - SETUP.md
- pubs/
  - RequestAccess.md

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: company-cluster-admins
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: company-team-cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: Group
 # DL SomeTeam : dl.someteam@company.com
 name: "asdfasdf-asdf-asdf-asdf-asdfasdfasdf"
```

Documentation vs Publications



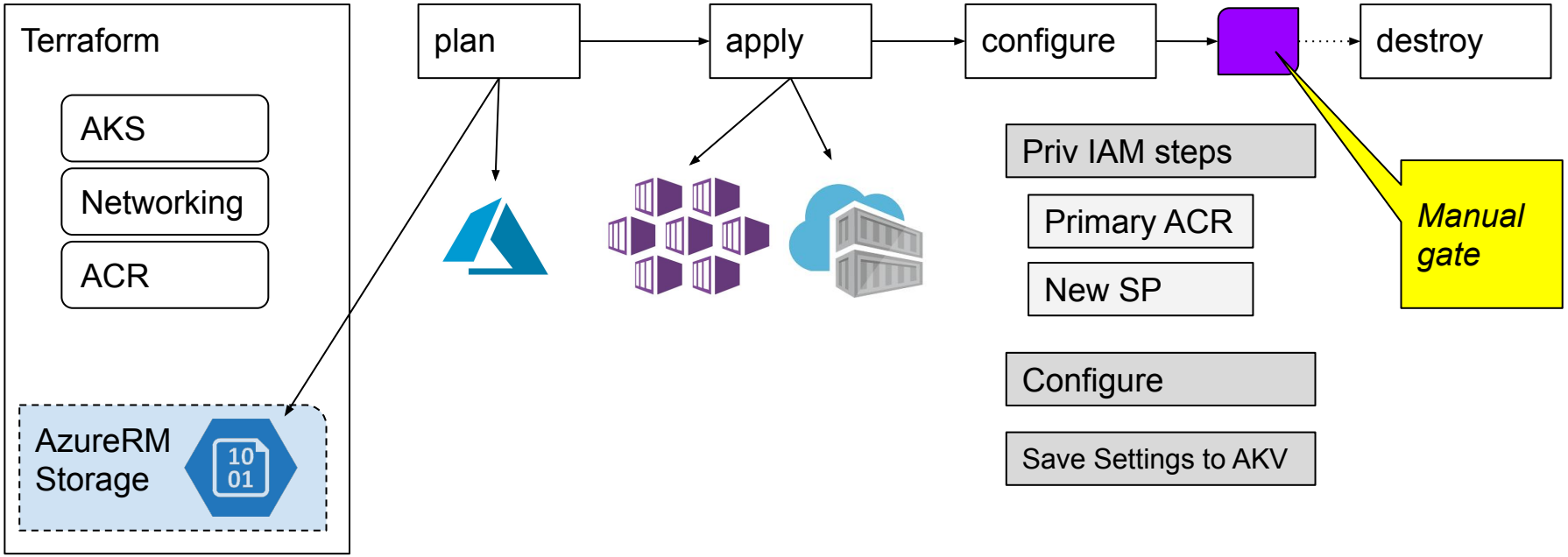
Docs:

- Runbooks
- How-to's
- Legacy one-offs
- Notes on Infra setup
- Team Meetings

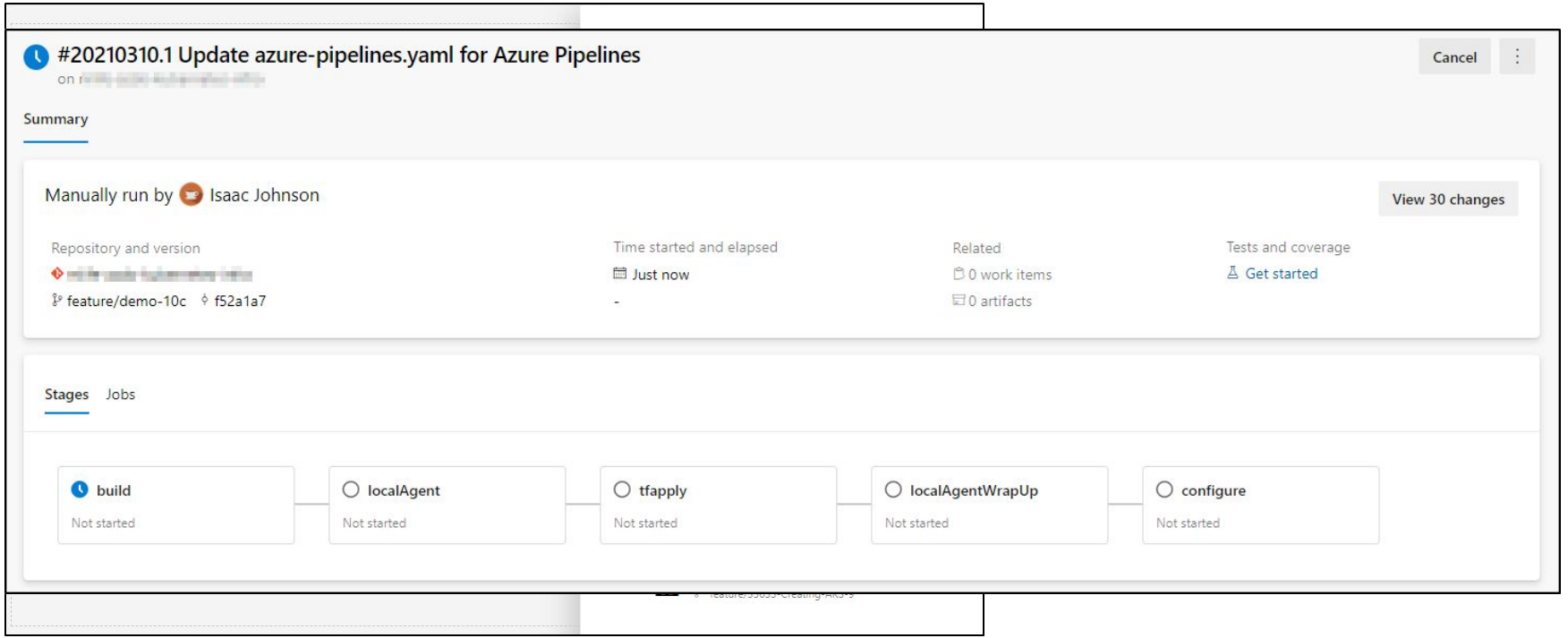
Pubs:

- How-to's for users
- FAQs
- REST Docs
- More information
- Status/Dashboards

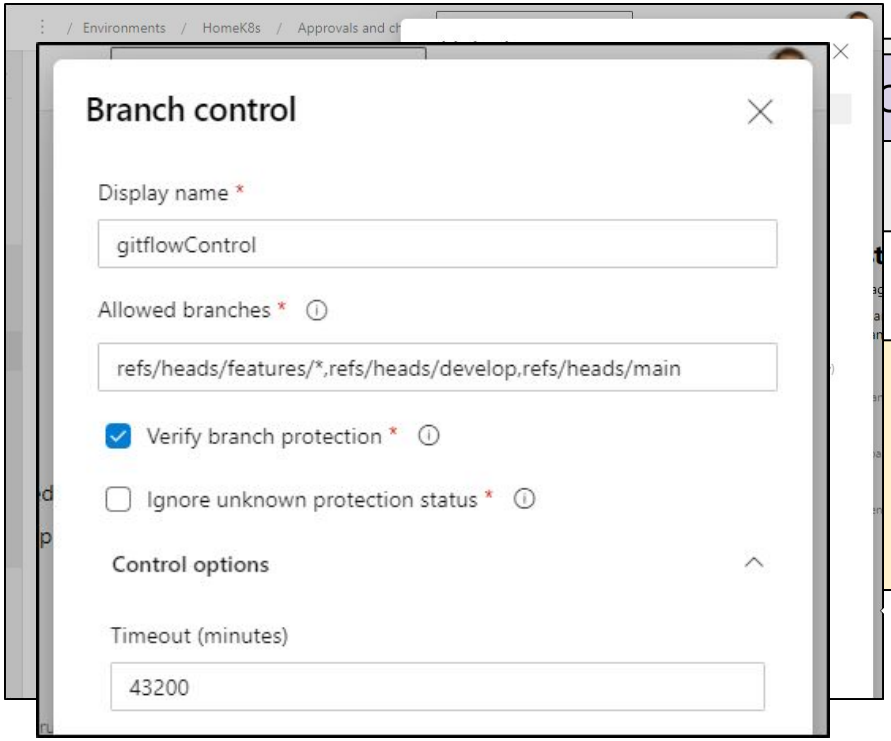
AKS ACR Process

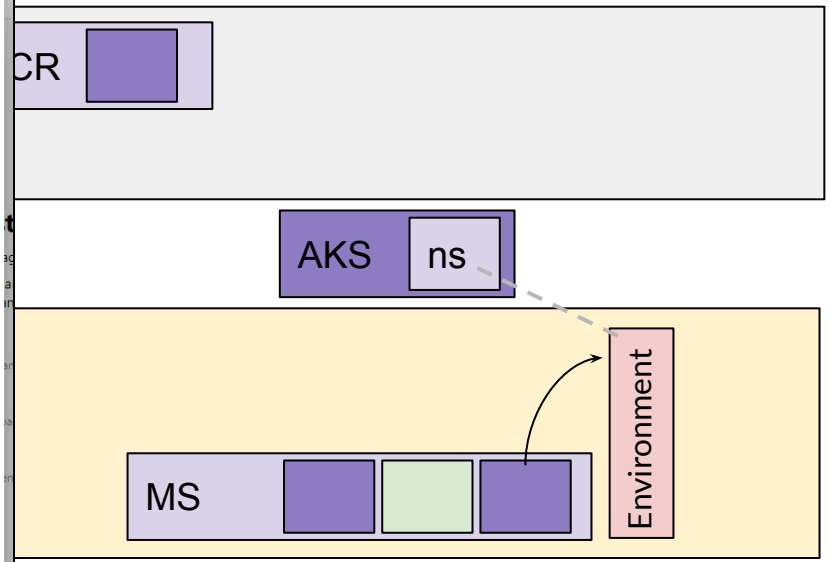


AKS ACR Process

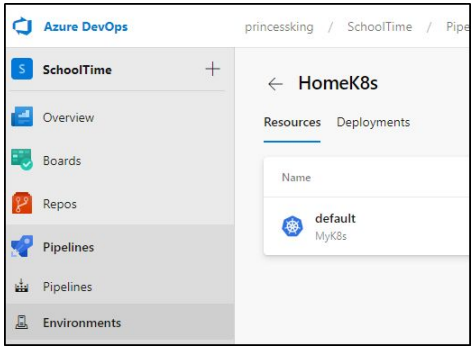


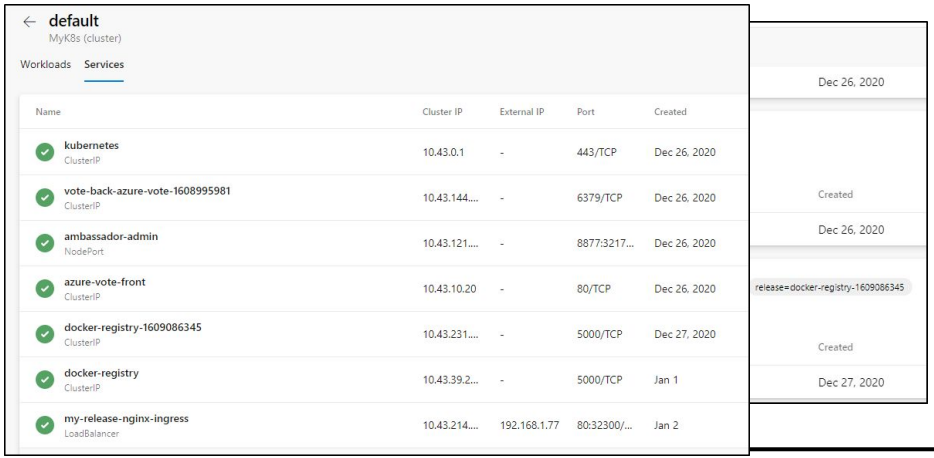
AKS ACR Process





Environment vs Service Connections





Expose just 1 namespace at a time

Created interactively

Require exposed management plane

Easy to view Workloads and Services, including those outside AzDO deploys

Developers need only to refer to environment (no need to know KV or access keys)

Considerations

- Kubernetes is not static: Be it EKS, GKE or AKS, “supported versions” in a region or zone keep changing - expect to have to upgrade.
- Consider IP Subnets when using Azure CNI or non-kubenet networking (every pod, node, etc takes an IP).
  - If you’ve exhausted your range, you may have to create a new Node Pool in a different subnet manually
- AKS with AAD uses your Access Token Lifetime from the Identity Platform
  - You may not be able to change the default 1 hour timeout

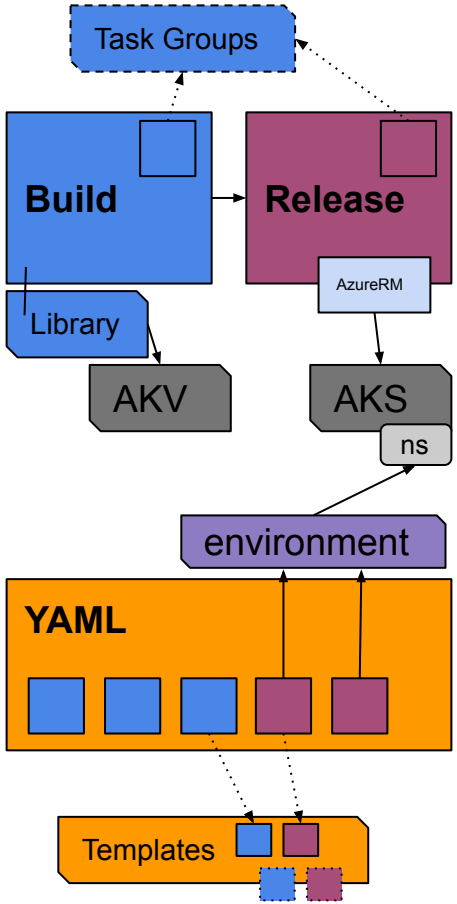
Microservice Patterns

Scaling Microservices

- Requirements
- Migration to YAML
- Patterns
- Process Overview
- Scaling to multi-cloud

Requirements

- Support Java and Dotnet microservices
  - Might have different versions of java and .NET
- Auto generate .NET and Java client bindings from Swagger (swagger gen)
- Deploy using YAML and/or Helm
- Use patterns for Sonar exclusions, Unit Tests



Phased Migration (Classic To YAML)

ClassicUI / AKS

- Task Groups
- Libraries for AKV
- Build Pipelines created Pipeline Artifacts (drop.zip) handed off to Release Pipelines
- Release Pipelines Gates on Branches
- Leveraged Kubernetes Tasks with AKV

YAML / Environments

- Multi-Stage YAML
- leveraged Templates from controlled project
- Yaml/Helm to “Environment” which could be gated on branch
- Secrets directly with AKV task
- YAML templates for WI Automation for onboarding

Task Groups vs YAML Templates (Classic to YAML)

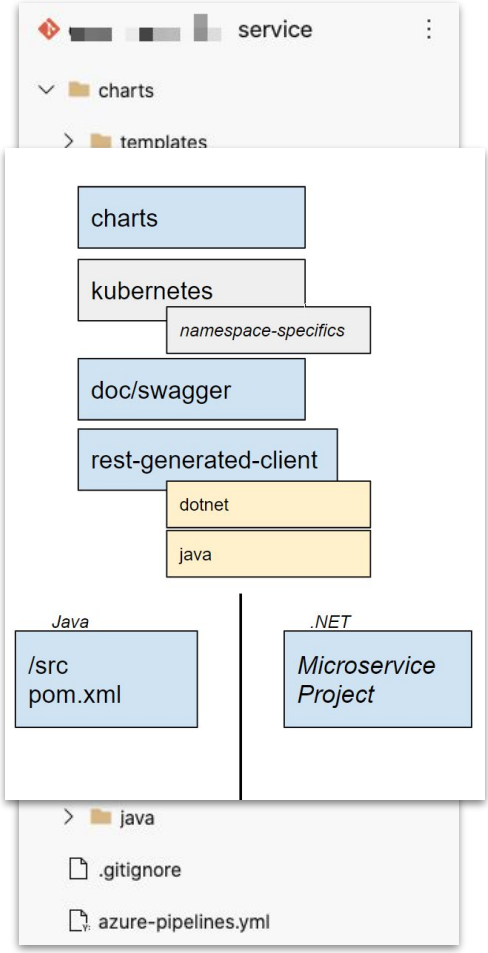
```yaml
- name: myStep
  type: step
  default:
    script: echo my step
- name: mySteplist
  type: stepList
  default:
    - script: echo step one
    - script: echo step two

trigger: none

jobs:
- job: stepList
  steps: ${{ parameters.mySteplist }}
- job: myStep
  steps:
    - ${{ parameters.myStep }}
```

Task Groups

- One version, but with history
- Task Group permissions for all task groups
- Cannot control parameters (auto detected)
  - Can set default and description
- Project Bound
- Visual Editor
- Right click to create (easy)

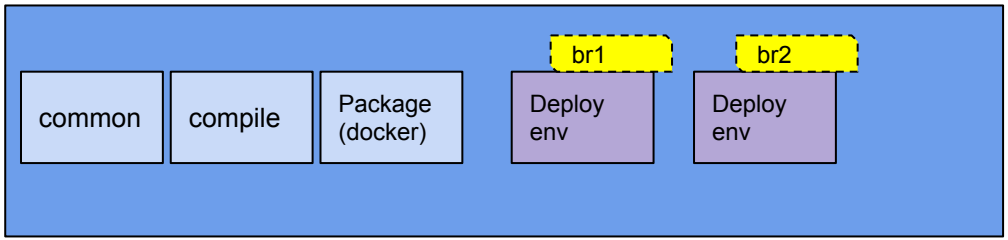
YAML Templates

- Multiple Branches, history
- PR policies and easy to contribute
- Can control parameters and defaults
- Extends
- Conditional Logic
- Can be used for stage, job, steps, etc
- Pull from any git provider and local
- No visual editor*

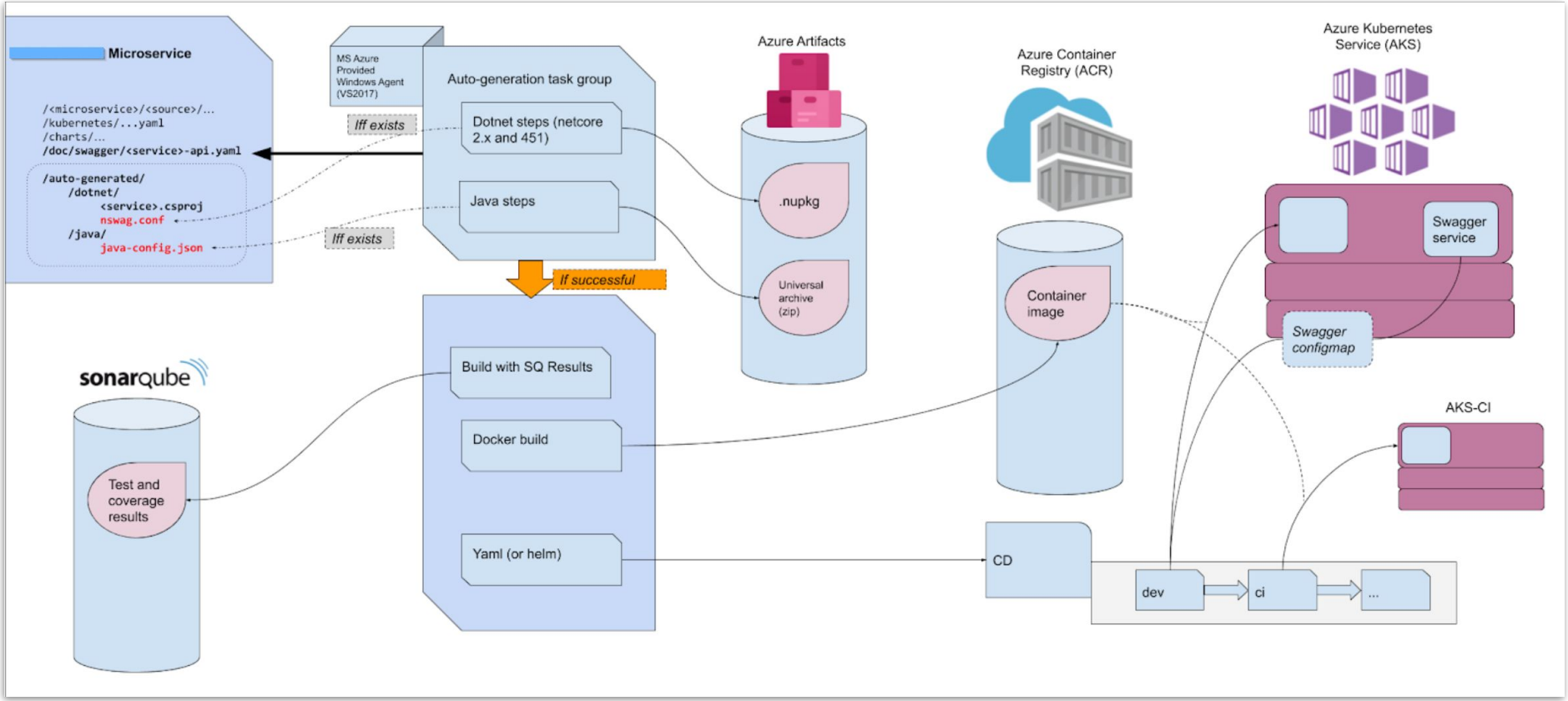


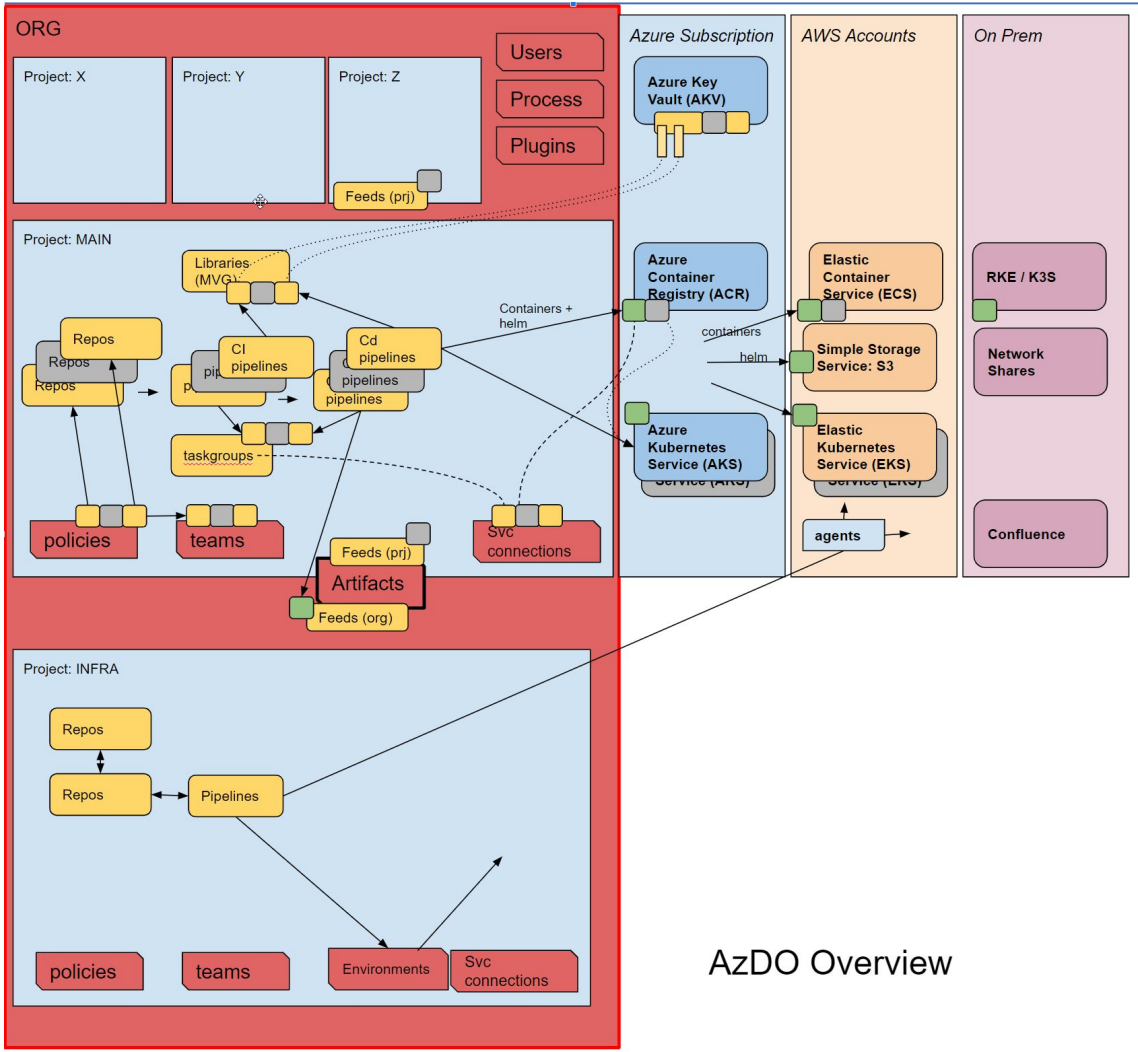
Makeup of a Microservice

1. Helm Chart (and or/ Yaml files)
2. Service itself
3. Swagger docs (used for documentation and auto-gen)
4. Rest-generated-client (used for auto-gen specifics)
5. Pipeline file



Full Process : Overview





AzDO Overview

Multi-Cloud

Following patterns allowed us to scale

Charts/Containers/Helm (green) are the same over multiple environments

INFRA team creating agents in restricted environments (sharing them to MAIN)

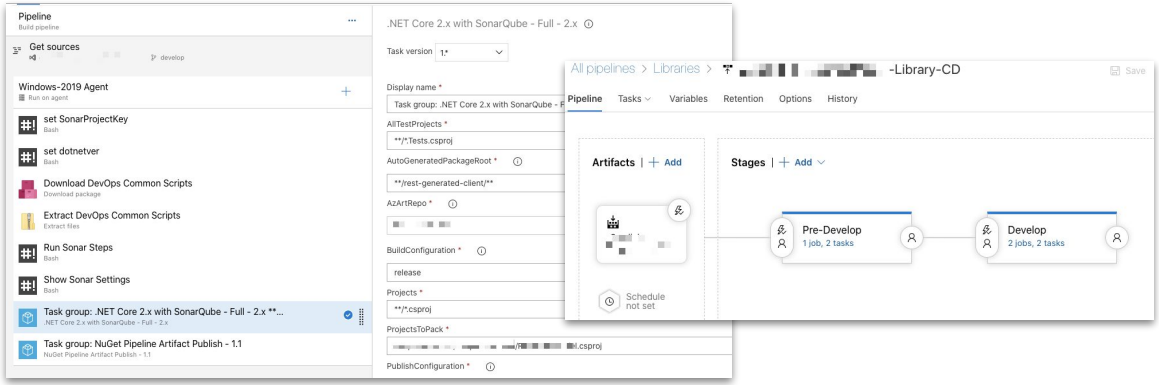
Key differences

Classic UI

- Still used with Test Plans
- Easy Graphic Interface
- Managed by “Build Team”
- **Control at Release Pipelines/Gates**

YAML Templates

- All modern CICD use YAML
- Empowers Developers to own pipelines and contribute to common Templates
- **Control at Environments**



How do my developers learn about Azure DevOps?

How do we inform but also enforce Patterns?

What is our cost model? Do we charge back? Can we leverage VSE?

Where is our compute? Where do we deploy?

How do we manage identities?

Considerations

- Empowering Developers with YAML requires trust
- Requiring managed agents creates choke points
  - VMSS and Azure Pipelines scale
- Library (AKV / Group Variables) can create unnecessary abstraction
  - Directly access with AKV
  - Minimize blast radius by more narrowly focused Vaults
  - Library requires manual selection/mapping
- Minimize use of “Build Variables” settable at build time
  - Don’t try to make AzDO into a utility Jenkins job

Can I rely on a single cloud PaaS?

What are my migration considerations if the offering becomes no longer viable?

What are my methods of control and auditing?

What if we succeed? How does Day 2 look? How do we scale? What do those costs look like?

Growth Areas

- Infrastructure
  - The right way to expose secrets to Pods
  - Service Mesh Options (Istio, Consul, etc)
  - Ingress (Istio, Nginx, Ambassador, etc)
  - Secrets Providers (Hashi Vault, or AKV)
    - Csi-secrets-store-provider-azure for AKS with managed identity
- Promotions
  - Proper way to promote releases
    - Especially interdependent or loosely coupled to a data layer
  - Proper way to validate and test

Thank you / Questions

Slides:

https://bit.ly/2NnEeZT

- https://freshbrewed.science/OSN2021/OSN_+Full+Stack+CICD+of+Kubernetes+Microservices+using+DevOps+and+IaC.pdf
- https://freshbrewed.science/OSN2021/OSN_+Full+Stack+CICD+of+Kubernetes+Microservices+using+DevOps+and+IaC.pptx

Blog: https://freshbrewed.science