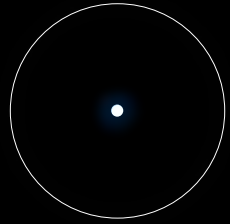


MAXIMIZE CI/CD EFFICIENCY: REUSABLE TEMPLATES WITH AZURE PIPELINES

RELEASING SOFTWARE IS TOO OFTEN AN ART; IT
SHOULD BE AN ENGINEERING DISCIPLINE

(DAVID FARLEY)





TRAVIS GOSSELIN

DISTINGUISHED SOFTWARE ENGINEER

DEVELOPER EXPERIENCE 

travigosselin.com 

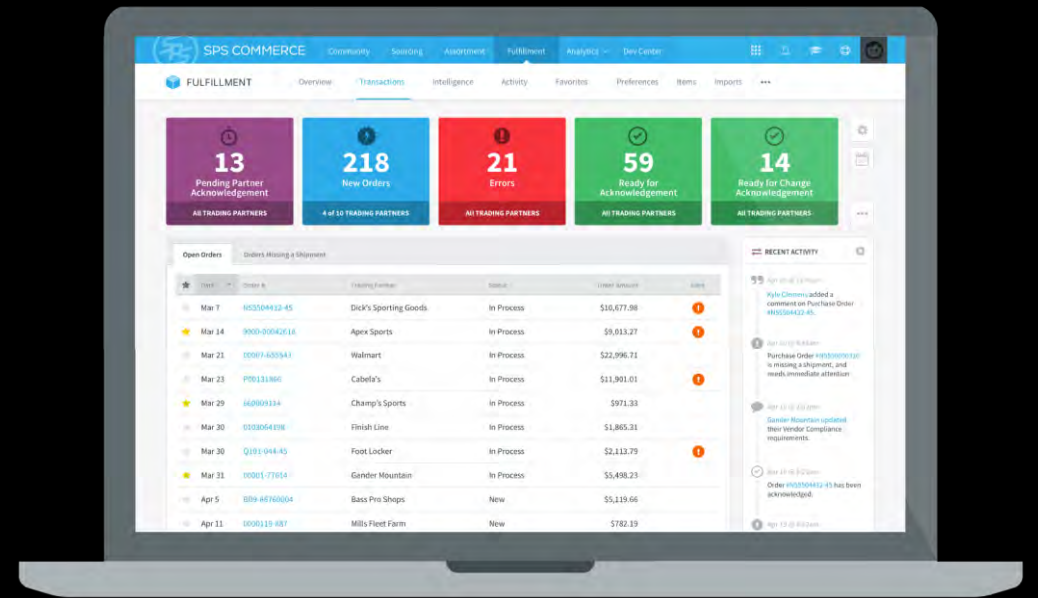
linkedin.com/in/travigosselin 

@travisjgosselin 



SPS COMMERCE

INFINITE RETAIL POWER™



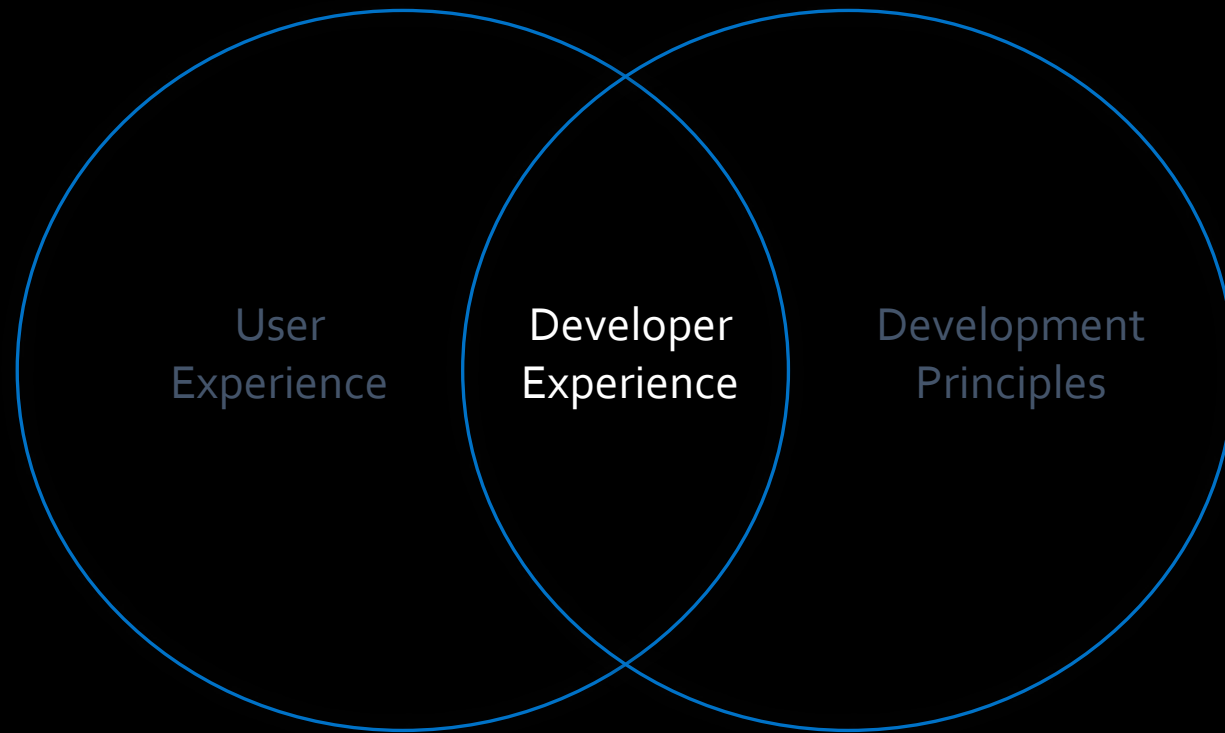


Developer Experience is the activity of studying, improving and optimizing how developers get their work done.

theapplslab.com (2017)

DEVELOPER EXPERIENCE

WHAT IS THAT...EXACTLY?



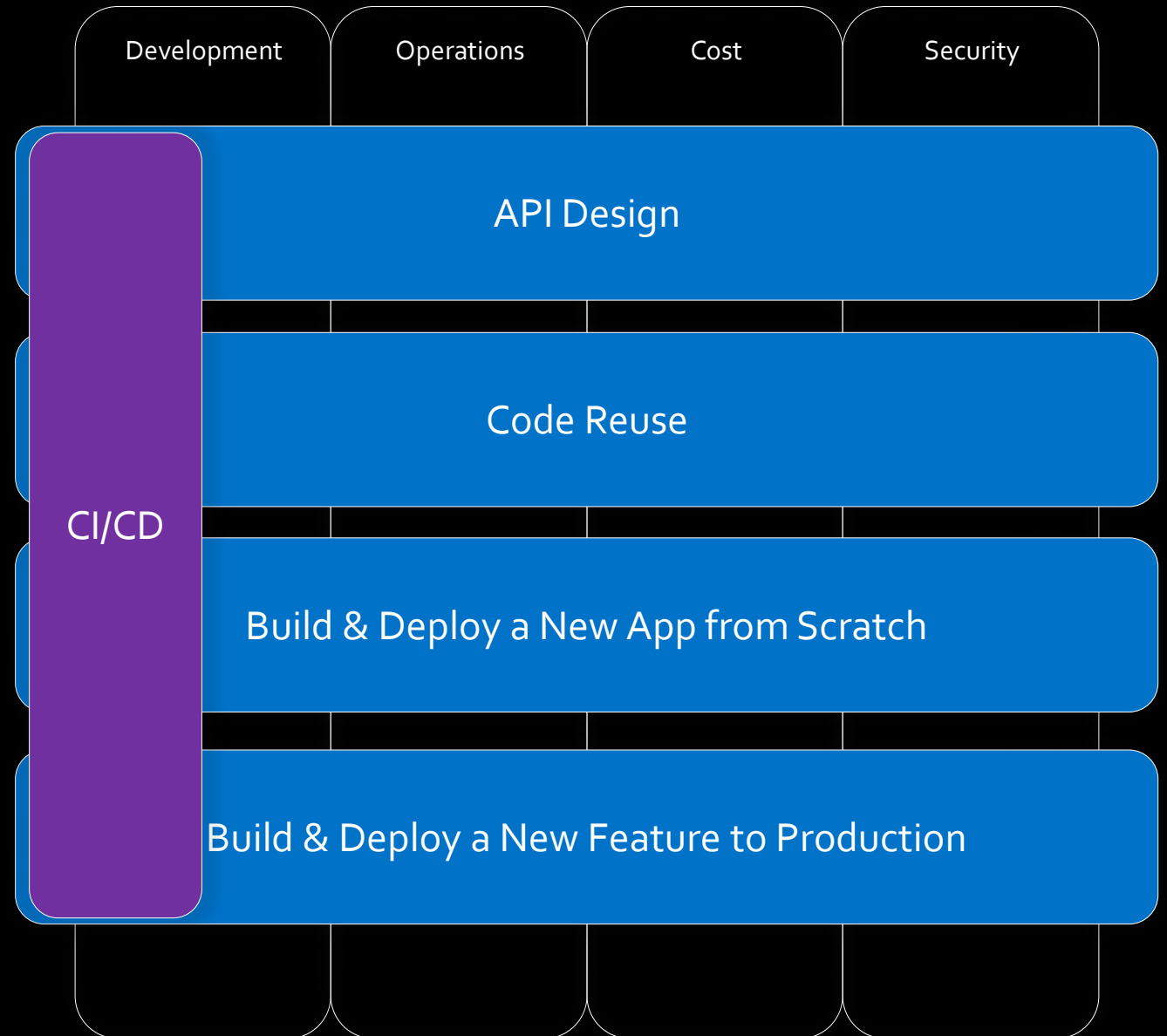
Developers work in rainforests, not planned gardens.

a16z.com

DEVELOPER EXPERIENCE: CAPABILITIES

IDENTIFIED HORIZONTAL FAST TRACKS TO BE
CURATED FOR MAXIMUM PRODUCTIVITY.

- Reduce toil
- Keep developers in “creative flow”
- Reduce feedback loop duration
- Reduce duplication
- Simplify
- Codify best practices



AZURE DEVOPS

Product & Services



Azure Boards

Work Items



Azure Repos

Version Control



Azure Pipelines

CI/CD



Azure Test Plans

Test Management



Azure Artifacts

Package Management

Visual Studio
Team Foundation Server

Visual Studio
Team Services

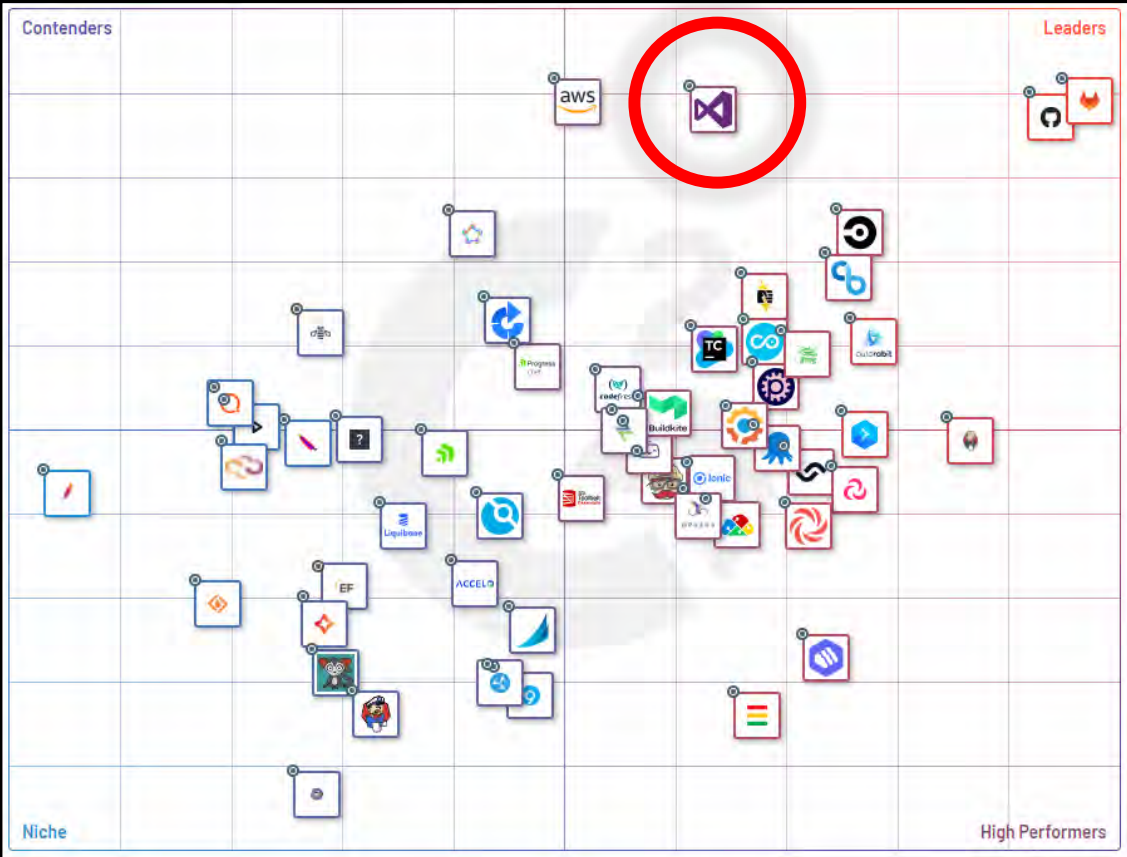
SaaS Product

Cloud & Self-Hosted Agents

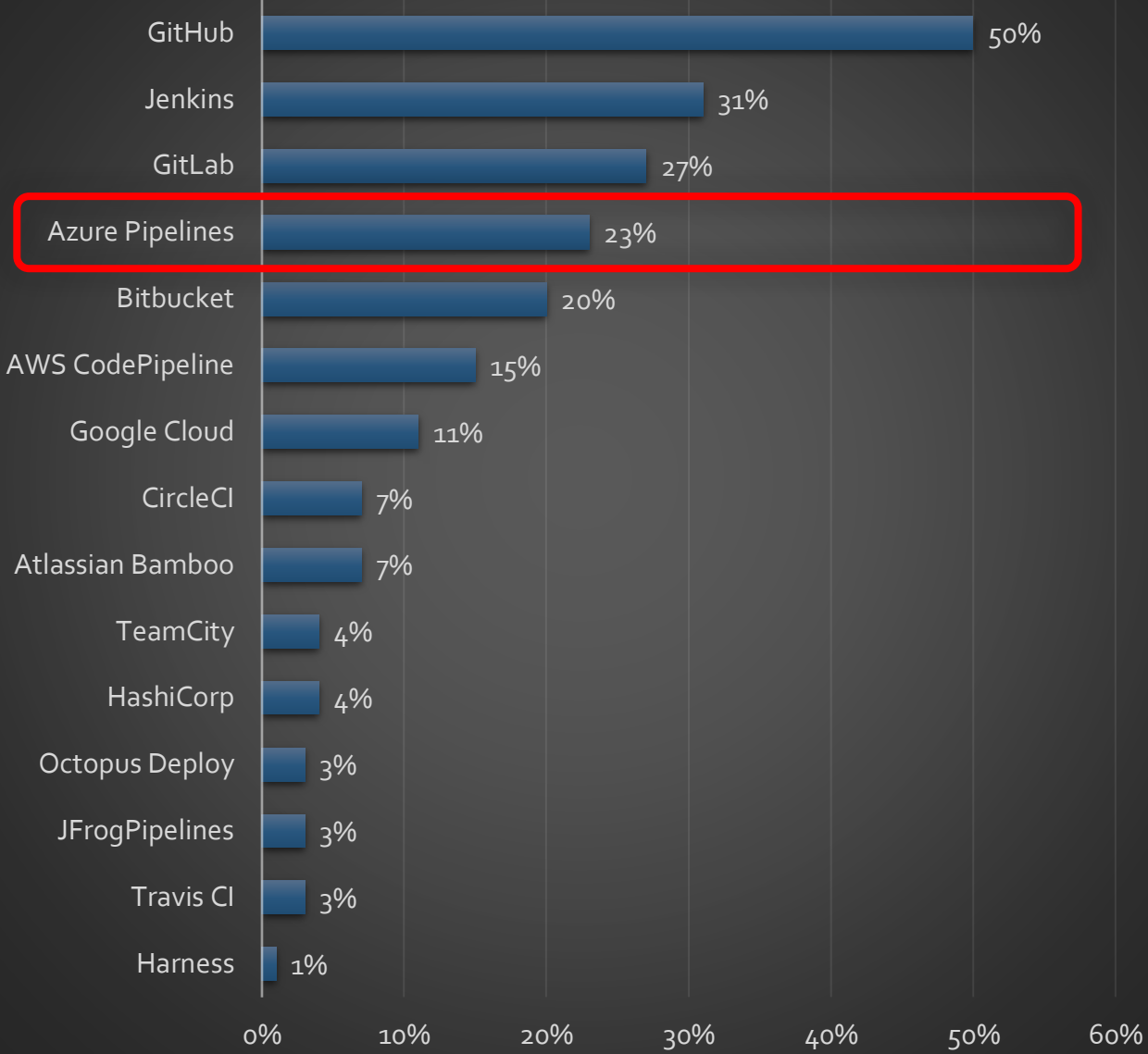
Marketplace

Product Name	Provider	Rating	Price
Test & Feedback	Microsoft	4.5/5 (73.4K)	FREE
Test Manager (Azure)	Microsoft	4.5/5 (19.9K)	FREE TRIAL
Test Case Explorer	Microsoft DevLabs	4.5/5 (11.3K)	FREE

Cloud Agnostic



CI/CD Tooling for APIs



AZURE PIPELINES

SPS COMMERCE

AZURE PIPELINES

OVERVIEW

#2.5.104 • Resolve issue in namespace. Run new

gates-api

SLACK || azure-pipelines-dev SERVICE || 76c5e2e2-99d9-4de8-a45d-8480860b219e TEAM || Site Reliability Engineering (SRE) PRODUCTION // CHGGMGMT-79359

This run is being retained as one of 50 recent runs by pipeline. View retention leases

Summary Tests Environments Artifactory Sauce Labs WhiteSource Bolt Build Report

Triggered by travisgosselin View 2 changes

Repositories 2 Time started and elapsed Related Tests and coverage

SPSPCommerce/azure-pipelines-gates-api, +1 Jan 5 at 4:50 PM 0 work items 100% passed

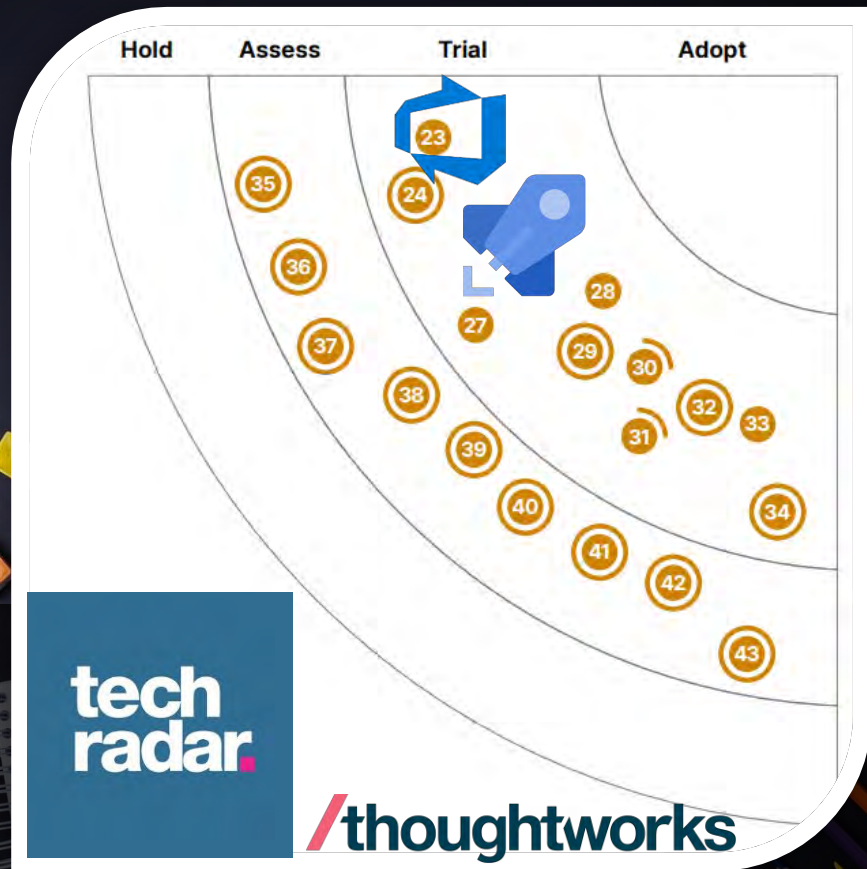
See Sources card for details 4h 34m 34s 2 published; 1 consumed Setup code coverage

YAML Multi-Stage Pipeline (Templating)

Stage	Jobs	Time	Checks
CONTEXT	1 job completed	16s	1 artifact
INITIALIZATION		13s	
COMPILE	3 jobs completed	6m 9s	100% tests passed, 1 artifact
INTEGRATION	4 jobs completed	9m 16s	2 checks passed
PROD APPROVAL	1 job completed	8s	1 check passed
PROD	4 jobs completed	11m 24s	4 checks passed

Sources

Repository	Branch / tag	Version	Related
SPSPCommerce/azure-pipelines-gates-api GitHub	main	6079033a	None



TEMPLATES

REUSABLE ORCHESTRATION

AZURE PIPELINES

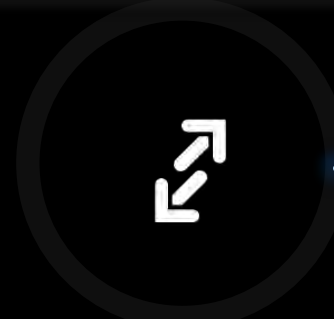
“ Templates let you define reusable content, logic, and parameters. Templates function in two ways. You can insert reusable content with a template or you can use a template to control what is allowed in a pipeline. ”

learn.microsoft.com



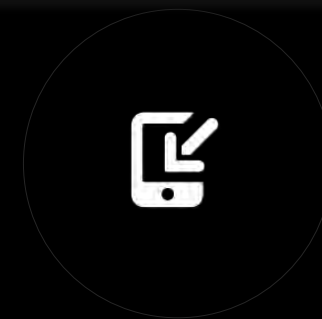
Anatomy

Tasks, Jobs, Stages,
Pipelines



Extend

Inheritance



Include

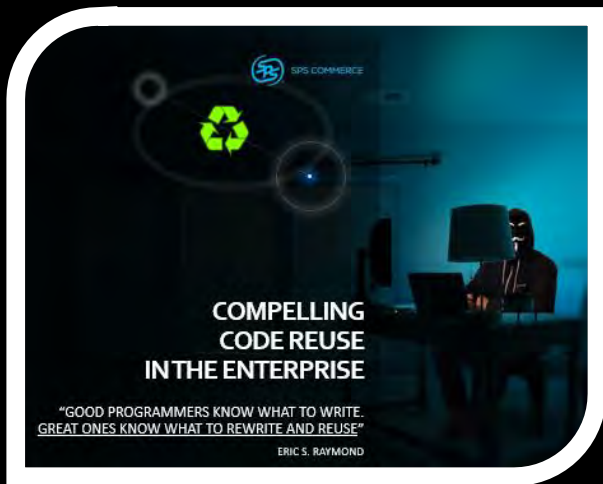
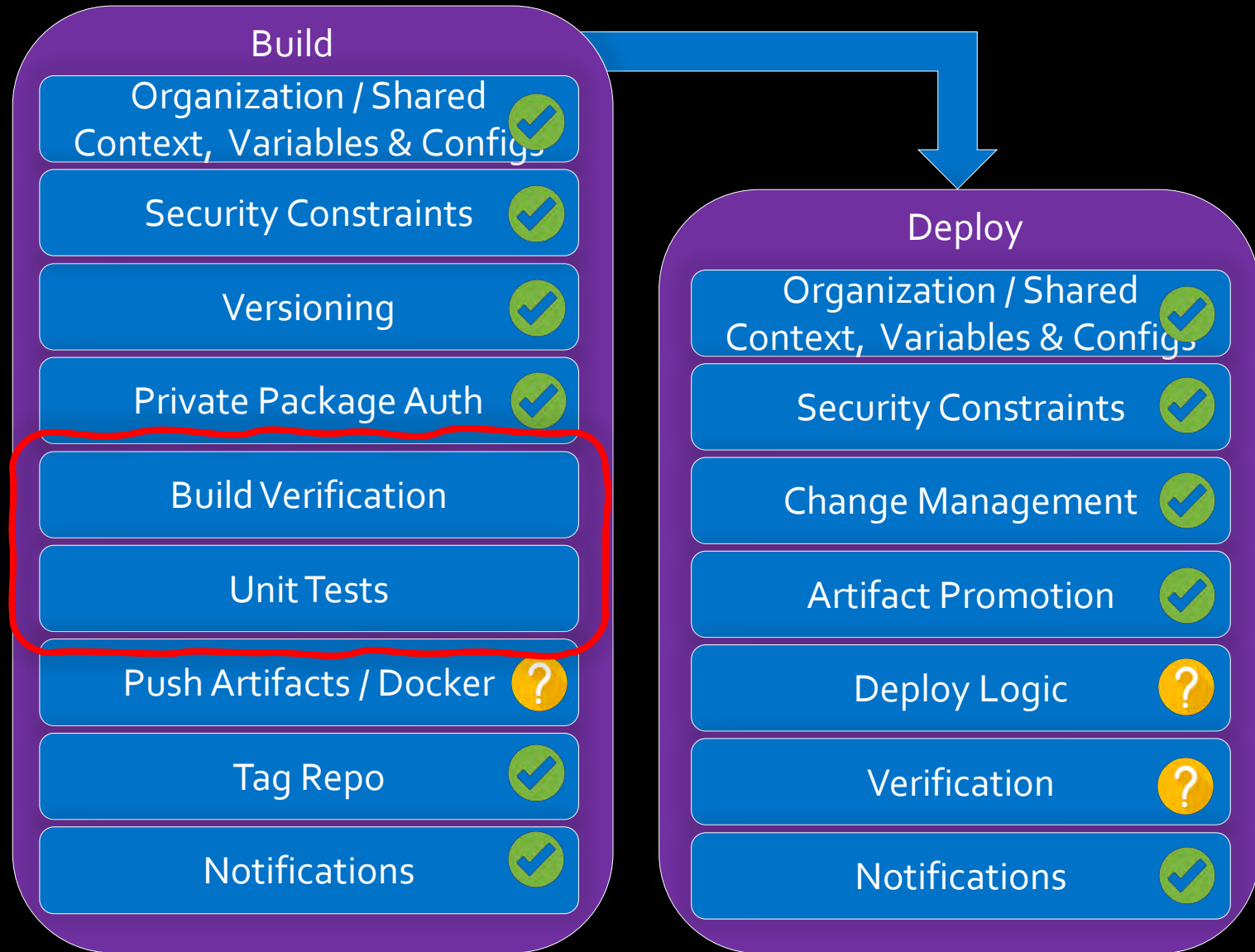
Composition



AZURE PIPELINE TEMPLATES

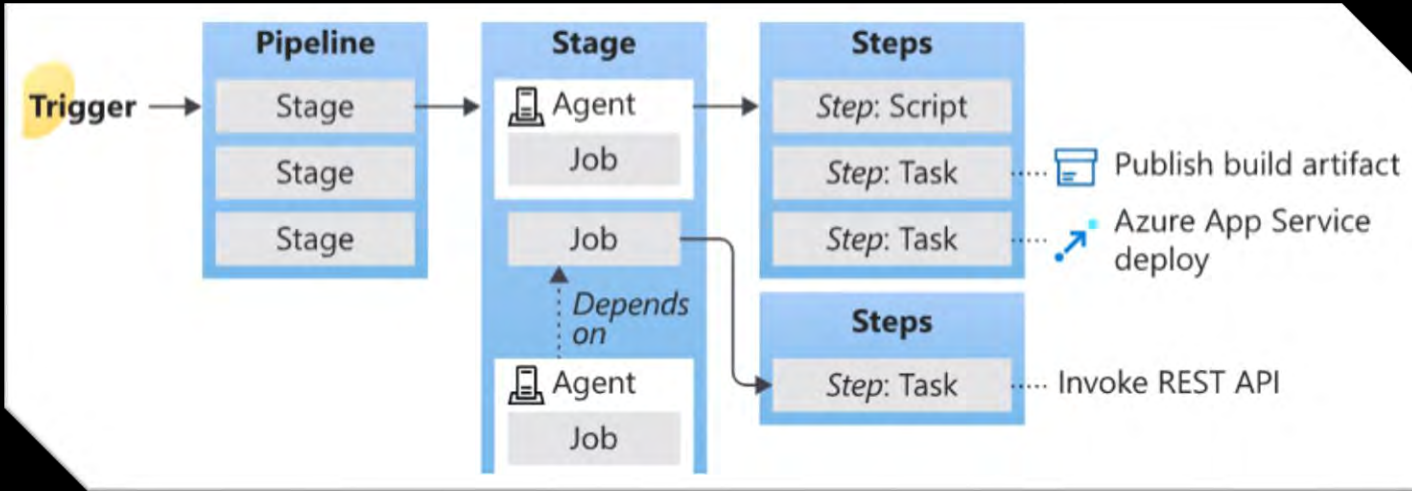
Functional Requirements

So much boilerplate!



AZURE PIPELINE TEMPLATES

Pipeline Anatomy



```
trigger:
- main
```

```
stages:
- stage: buildStage
  jobs:
  - job: buildJob
    steps:
    - bash: echo "Hello Step 1"
    - bash: echo "Hello Step 2"
  - job: testJob
    steps:
    - bash: echo "testing"
```

Stage Sequence (bracketed on the left)

Jobs Parallel (bracketed on the right)

```
- stage: deployStage
  jobs:
  - deployment: deployjob
    environment: development
    strategy:
      runOnce: DeploymentJob
    deploy:
      steps:
      - bash: echo "Deploying"
```

AZURE PIPELINE TEMPLATES

Template Basics

parameters:

- name: name
type: string
default: travis
- name: environment
type: string
values:
 - dev
 - prod
- name: requiresApproval
type: boolean
default: true

stages:

- stage: deploy_\${{ parameters.environment }}
displayName: Deploy \${{ upper(parameters.environment) }}
- jobs:
 - job: execute
steps:
 - bash: echo "Hello \${{ parameters.name }}"

TYPES: string, number, boolean

Required or not

Restricted Enumerations

Stages, Jobs, Steps, Variables

Functions

eq(variables.letters, 'ABC')

gt(5,2)

length('travis')

or(eq(1, 1), eq(2, 3))

startsWith('ABCDE', 'AB')

not(eq(1, 2))

AZURE PIPELINE TEMPLATES

Template Expressions

parameters:

- name: environments
- type: object
- default:
 - dev
 - prod

COMPLEXTYPES

- name: complexEnvironments
- type: object
- default :
 - name: dev
 - approvalRequired: false
 - approvers:
 - user1
 - user2
 - name: prod
 - approvalRequired: true
 - approvers:
 - user3
 - user4

step, stepList

job, jobList

stage, stageList

deployment,
deploymentList

- name: deploySteps
- type: stepList
- default:
 - bash: echo "Hello Travis"

stages:

- stage: deploy
- jobs:
 - job: execute
 - steps:

- \${{ parameters.deploySteps }}

- bash: echo "Hello Again!"

- \${{ each env in parameters.environments }}:

- bash: echo \${{ env }}

- \${{ each env in parameters.complexEnvironments }}:

- \${{ if eq(env.approvalRequired, true) }}:

- \${{ each approver in env.approvers }}:

- bash: echo "hello \${{ approver }}"

- \${{ else }}:

- bash: echo "no approval for \${{ env.name }}"

AZURE PIPELINE TEMPLATES

Variable Templates

```
#-- dev.variables.yml
variables:
- name: environmentName
  value: dev
- name: baseUrl
  value: https://dev.api.platform.com
```

```
#-- prod.variables.yml
variables:
- name: environmentName
  value: prod
- name: baseUrl
  value: https://api.platform.com
```

```
#-- deploy.template.yml
parameters:
- name: environment
  type: string
  values:
  - dev
  - prod

stages:
- stage: deploy
  variables:
  - template: ${ parameters.environment }.variables.yml

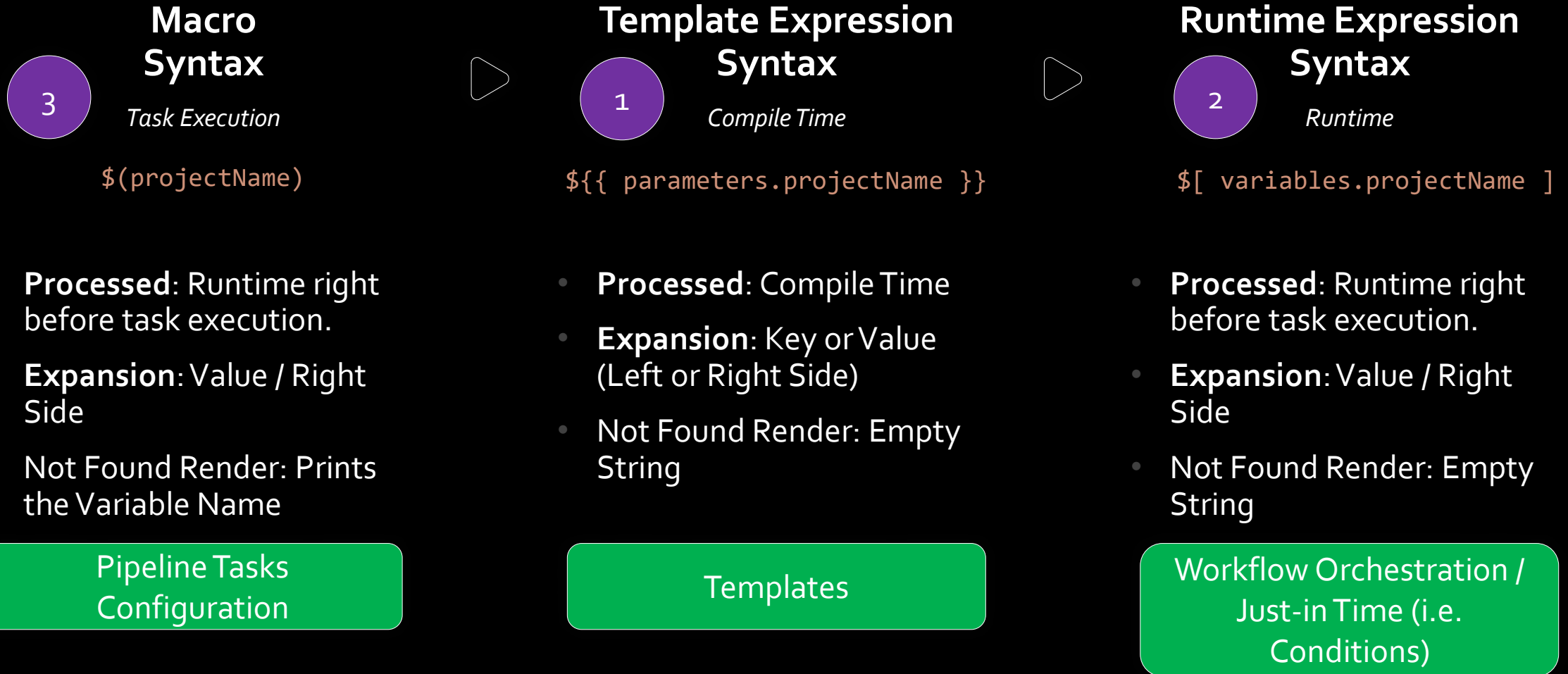
jobs:
- job: execute
  steps:
  - bash: echo "$(environmentName) at $(baseUrl)"
```

Template Import

Variable Syntax?

AZURE PIPELINE TEMPLATES

Variable Syntax



isMain: `$(eq(variables['Build.SourceBranch'], 'refs/heads/main'))`

AZURE PIPELINE TEMPLATES

Including a Template Locally

```
#-- azure-pipelines.yml
...
stages:
- stage: build
  jobs:
  - job: execute
    steps:
    - bash: echo "Hello build"
    - template: deploy/build-prep.job.template.yml
  - template: deploy/deploy.stage.template.yml
    parameters:
      environment: dev
  - template: deploy/deploy.stage.template.yml
    parameters:
      environment: prod
      requireApproval: true
```

Naming Convention
Suffix - Job, Stage, Step,
Variables



my-repository

- azure-pipelines.yml
- > deploy
 - deploy.stage.template.yml
 - build-prep.job.template.yml
 - variables.template.yml

Relative Paths – to File Including It

Templates in Parameters to Other Templates

Max 20 Levels of Template Nesting

100 Separate YAML Files - Direct or Indirect

Max 10 MB Memory Parsing (1 – 2 MB YAML)

AZURE PIPELINE TEMPLATES

Including a Template Remotely

```
##-- azure-pipelines.yml
...
resources:
  repositories:
    - repository: templates
      type: github
      name: org/azure-pipelines-templates
      endpoint: github-connection
      ref: refs/heads/main
  stages:
    - stage: build
      jobs:
        - job: execute
          steps:
            - bash: echo "Hello build"
            - template: deploy/build-prep.job.template.yml@self
    - template: deploy.stage.template.yml@templates
      parameters:
        environment: dev
    - template: deploy.stage.template.yml@templates
      parameters:
        environment: prod
        requireApproval: true
```

Versioning?



azure-pipelines-templates (@templates)

- deploy.stage.template.yml
- > environments
 - dev.variables.template.yml
 - prod.variables.template.yml



my-repository (@self)

- azure-pipelines.yml
- > deploy
 - build-prep.job.template.yml

Changes Impact All Organizational Pipelines



AZURE PIPELINE TEMPLATES

Organizational Template Versioning

Branch-Based

```
resources:  
  repositories:  
    - repository: templates  
      type: github  
      name: org/templates  
      endpoint: github-connection  
      ref: refs/heads/v1
```

template: `deploy.stage.yml@templates`

Coupled Template Versions

Non-Immutable Usage

Explicit Reference

Commit / Tag-Based

```
resources:  
  repositories:  
    - repository: templates  
      type: github  
      name: org/templates  
      endpoint: github-connection  
      ref: refs/tags/v1.2  
      # ref: f125bdd
```

template: `deploy.stage.yml@templates`

Coupled Template Versions

Immutable Usage

Explicit Reference

Treat it like API Versioning

Naming Convention

```
resources:  
  repositories:  
    - repository: templates  
      type: github  
      name: org/templates  
      endpoint: github-connection
```

template: `deploy.stage.v1.yml@templates`

Decoupled Template Versions

Non-Immutable Usage

No Explicit Reference

Simpler Git Maintenance

AZURE PIPELINE TEMPLATES

Extending a Template

```
## azure-pipelines.yml
```

```
...
resources:
  repositories:
    - repository: templates
      type: github
      name: org/azure-pipelines-templates
      endpoint: github-connection
```

```
extends:
  template: base.template.v1.yml@templates
parameters:
  stages:
    - stage: build
      jobs:
        - job: execute
          steps:
            - bash: echo "Hello build"
```

Extends Behavior

```
## base.template.v1.yml
```

```
parameters:
  - name: stages
    type: stageList

variables:
  - template: variables.v1.yml

stages:
  - stage: CONTEXT
    jobs:
      - template: context.job.v1.yml
      - ${{ parameters.stages }}
```

Enforce Stages, Jobs or Steps

Eliminate Boilerplate (i.e. Agent Pool)

Add Standard Variables

Add Required Jobs & Context



Pretty Powerful... but how does this help with security?

AZURE PIPELINE TEMPLATES

Extending a Template - Advanced

```
# -- base.template.v1.yml
parameters:
- name: stages
  type: stageList
- name: includeArtifacts
  type: boolean
  default: true

variables:
- template: variables.v1.yml

stages:
- stage: CONTEXT
  jobs:
  - template: context.job.v1.yml
...
```

```
...
# dynamic incoming stages
- ${{ each stage in parameters.stages }}:
  - ${{ each pair in stage }}:
    ${{ if ne(pair.key, 'jobs') }}:
      ${{ pair.key }}: ${{ pair.value }}
  jobs:
  - ${{ each job in stage.jobs }}:
    - ${{ each pair in job }}:
      ${{ if ne(pair.key, 'steps') }}:
        ${{ pair.key }}: ${{ pair.value }}
      ${{ if eq(pair.key, 'steps') }}:
        steps:
        - ${{ if eq(parameters.includeArtifacts, true) }}
          - download: current
            artifact: $(myArtifacts)
            displayName: download standard artifacts
        - ${{ each step in job.steps }}:
          - ${{ if contains(lower(step.value), lower('CmdLine')) }}:
              '{{ pair.value }}': error
          - ${{ else }}:
              - ${{ each stepPair in step }}:
                ${{ stepPair.key }}: ${{ stepPair.value }}
```

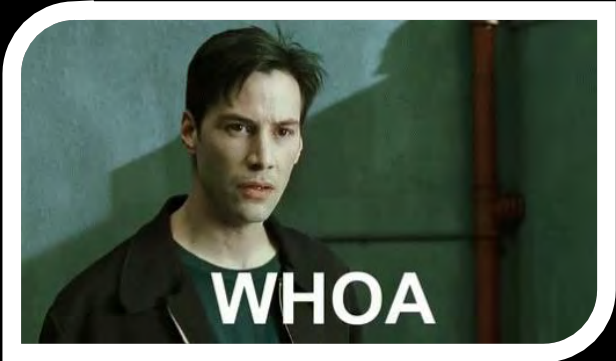
Disallow Syntax
(i.e. DeploymentJob)

Apply Defaults

Configurable Behavior

Download Required
Defaults / Context

Restrictions



AZURE PIPELINE TEMPLATES

Inheritance & Composition

Appropriate Coupling

Remove Undifferentiated Engineering / Boilerplate

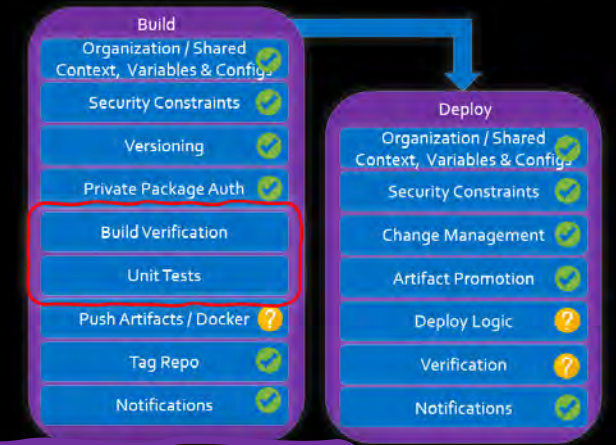
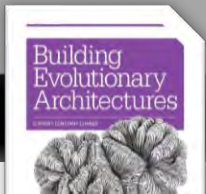
Supports Platform Engineering & IDPs

Great Developer Experience & Productivity

Evolutionary Architecture

An evolutionary architecture supports guided, incremental change across multiple dimensions.

Building Evolutionary Architectures



```
##-- azure-pipelines.yml
```

```
...
resources:
  repositories:
    - repository: templates
```

```
extends:
  template: base.template.v1.yml@templates
parameters:
  stages:
```

```
- template: context-and-version.stage.v1.yml@templates
- template: java-build.stage.v1.yml@templates
  parameters:
    dockerfile: Dockerfile
    customBuildCommand: ./mvnw --batch-mode clean compile
    postBuildSteps:
      - bash: echo "hello world!"
```

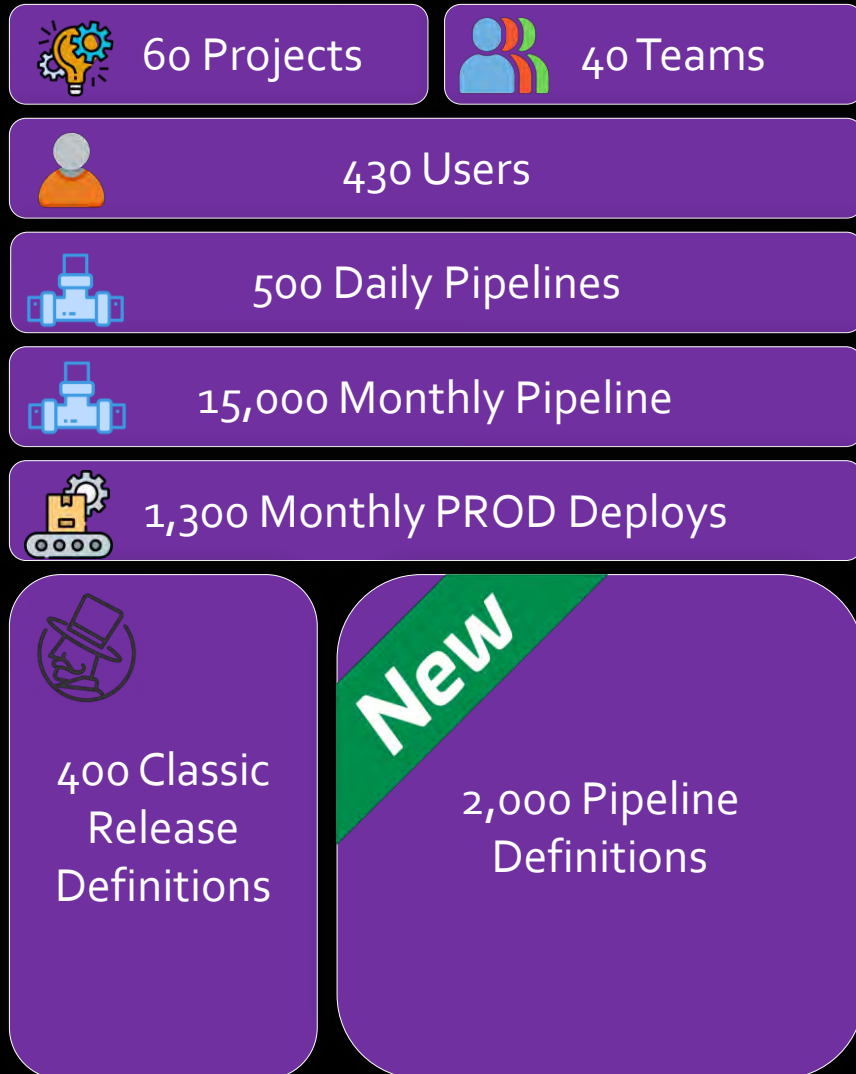
```
- template: deploy.stage.v1.yml@templates
  parameters:
    environment: dev
    tokenize: true
```

```
- template: deploy.stage.v1.yml@templates
  parameters:
    environment: prod
```




EXAMPLE ARCHITECTURE

SPS Commerce: Overview



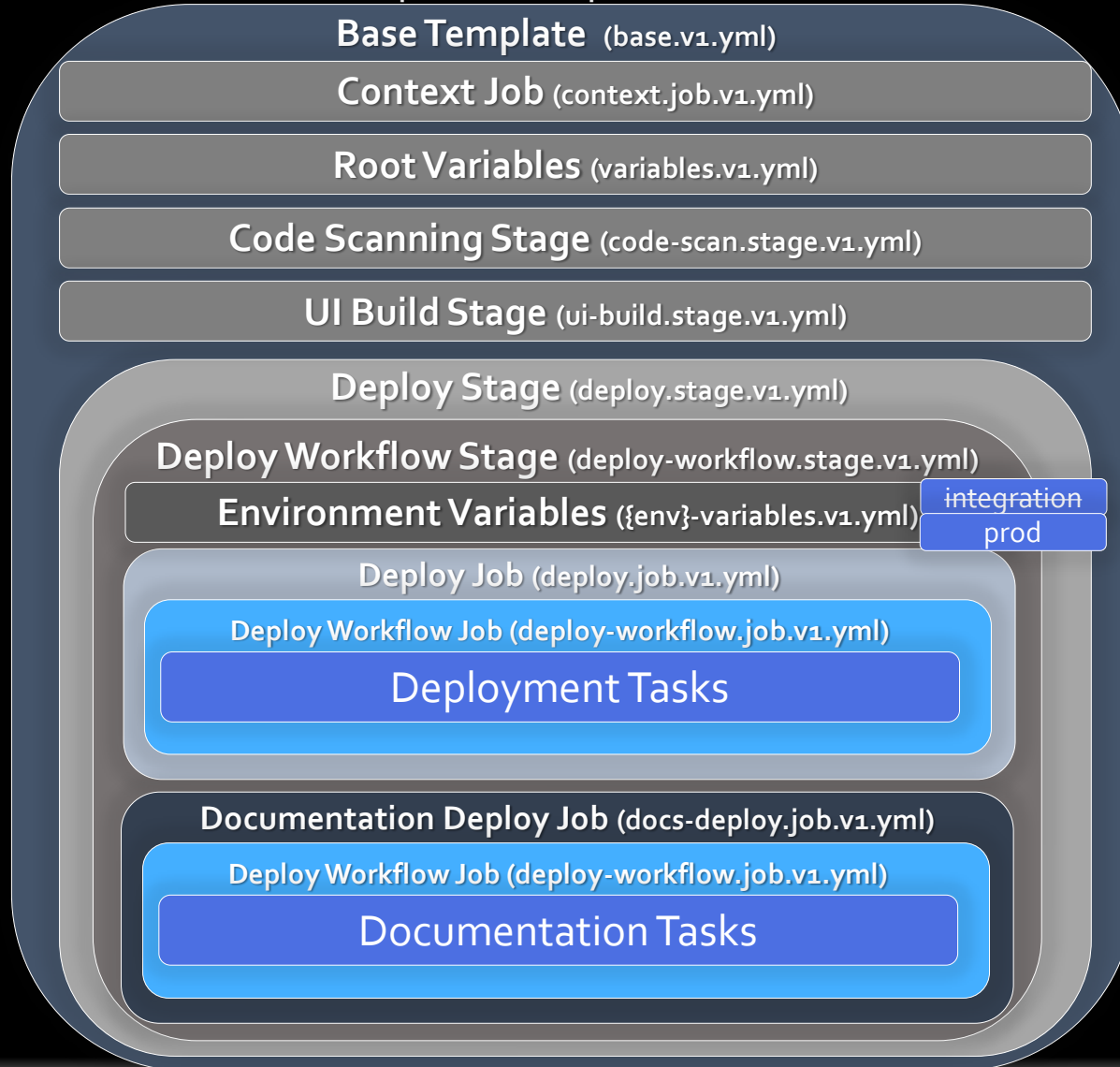
Design Goals, Requirements & Constraints

- Eliminate Team Management of Infrastructure
- Eliminate Redundant CI/CD Feature Implementation
- Single Tool for Building and Deploying
- Limited Click-Ops Required
- Flexible for Polyglot Ecosystem
- Custom Workflows, Orchestration & Governance
- Evolvable & Incremental Pipeline Architecture



EXAMPLE ARCHITECTURE

SPS Commerce: Template Composition



triggers:

...

resources:

...

extends:

```
template: base.v1.yml@templates
```

```
parameters:
```

```
bdpFile: .bdp
```

```
stages:
```

- template: ui-build.stage.v1.yml@templates
- template: code-scan.stage.v1.yml@templates

```
parameters:
```

```
language: javascript
```

- template: deploy.stage.v1.yml@templates
- ```
parameters:
```

```
environment: test
```

```
documentation: true
```

- template: deploy.stage.v1.yml@templates
- ```
parameters:
```

```
environment: prod
```

```
documentation: true
```

```
requireApproval: true
```



Don't Start with The Great Pyramid!

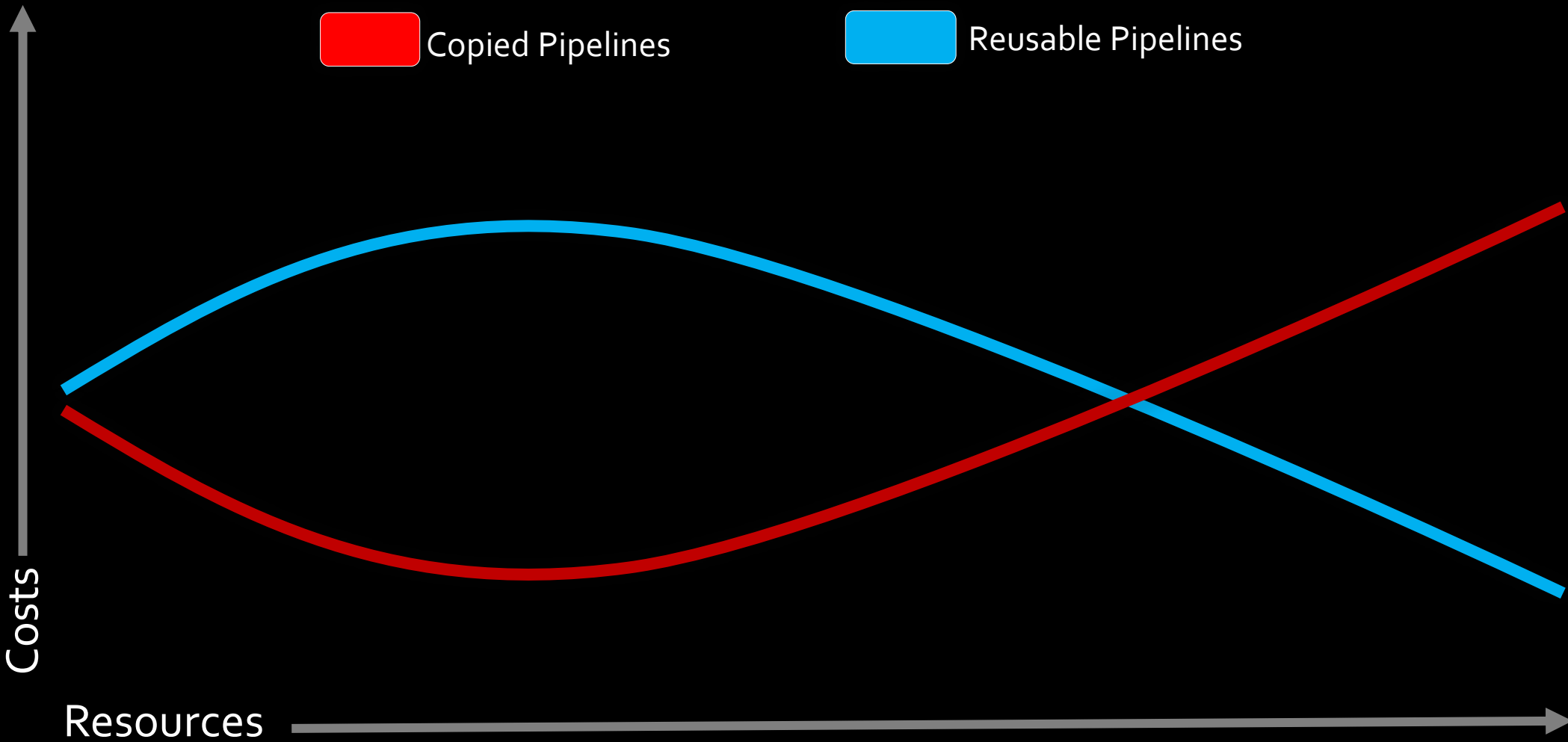
Build Small Composable Templates!

Weathering & Open Questions

- Abstracting too much knowledge?
- Retirement and Deprecation of Templates like Major API Versions?
- Taking advantage of other Deployment/Job Strategies...

DIMINISHING RETURNS

Duplicating Pipelines





"

When done well, software is invisible.

"

Bjarne Stroustrup

Azure Pipelines Invisibility Techniques

Templates


Environments, Approvals & Checks

Custom Tasks

MAXIMIZE CI/CD EFFICIENCY: REUSABLE TEMPLATES WITH AZURE PIPELINES



TRAVISGOSSELIN 

travisgosselin.com 

linkedin.com/in/travisgosselin 

@travisjgosselin 