THANK YOU TO OUR SPONSORS

Presented by Microsoft

Farm Credit Services of America

Union Pacific

Physicians Mutual

Google

Mutual of Omaha

Kiewit

Medical Solutions

fnts

BlueCross BlueShield Nebraska

DELIVER

OPPD Omaha Public Power District

csg INTERNATIONAL

BELLEVUE UNIVERSITY

Tigerpaw

valmont

PARAGON IT Professionals

TEAM Software

Multi-stage YAML Pipelines with Azure DevOps

VAIBHAV GUJRAL
MICROSOFT MVP - AZURE

About Me

- Cloud Architect at Kiewit Corp
- Microsoft Most Valuable Professional (MVP) – Azure
- Microsoft Certified Trainer (MCT)
- Microsoft Certified Azure Solutions Architect Expert
- Organizer, Omaha Azure User Group
- http://www.vaibhavgujral.com
- @vabgujral
- linkedin.com/in/vaibhavgujral/
- #AzureHeroes Community & Content Hero
- Listed among the “Top 50 Microsoft Azure Blogs, Websites & Influencers in 2020”

Agenda

- Azure Pipelines
- YAML Schema
  - Triggers
  - Stages
  - Jobs
  - Tasks
  - Dependencies and Conditions
  - Templates
- Classic vs YAML pipelines
- Q&A

Azure DevOps

Azure Boards

Deliver value to your users faster using proven agile tools to plan, track, and discuss work across your teams.

Azure Pipelines

Build, test, and deploy with CI/CD that works with any language, platform, and cloud. Connect to GitHub or any other Git provider and deploy continuously.

Azure Repos

Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.

Azure Test Plans

Test and ship with confidence using manual and exploratory testing tools.

Azure Artifacts

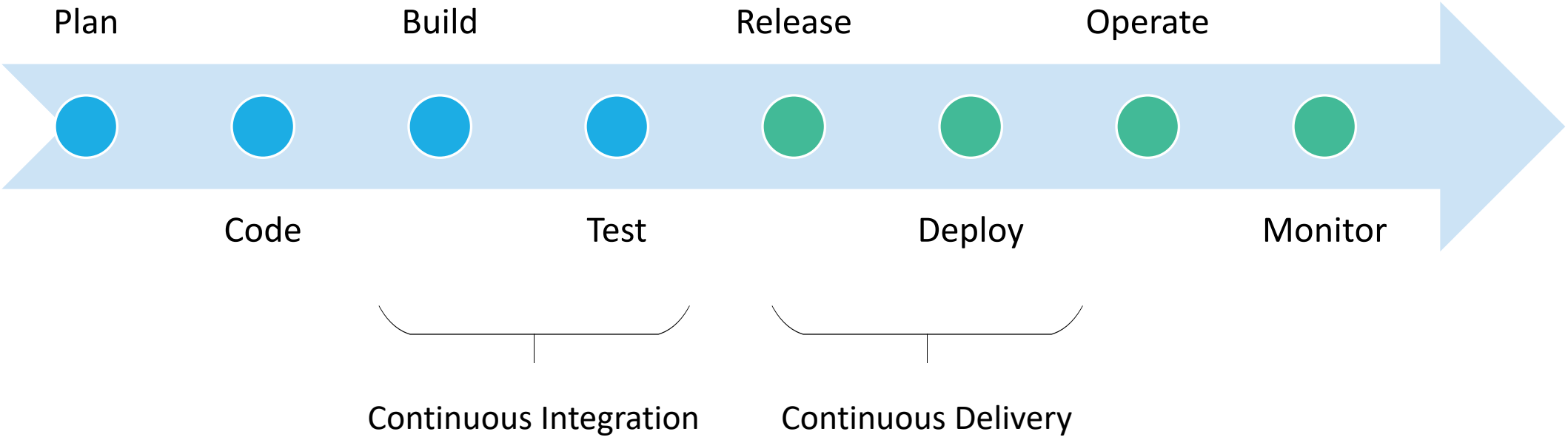
Create, host, and share packages with your team, and add artifacts to your CI/CD pipelines with a single click.

Extensions Marketplace

Access extensions from Slack to SonarCloud to 1,000 other apps and services—built by the community.

Continuous Integration (CI) and Continuous Deployment (CD)

Continuous Integration refer to the process of constantly and consistently testing and building code at every check-in, and continuous delivery refers to the process of shipping the code regularly to any target.



Continuous Integration (CI) and Continuous Deployment (CD)

Continuous Integration refer to the process of constantly and consistently testing and building code at every check-in, and continuous delivery refers to the process of shipping the code regularly to any target.

Continuous integration (CI)	Continuous delivery (CD)
- Increase code coverage - Build faster by splitting test and build runs - Automatically ensure you don't ship broken code - Run tests continually.	- Automatically deploy code to production - Ensure deployment targets have latest code - Use tested code from CI process.

Azure Pipelines

Azure pipeline is a cloud service that offers continuous integration and continuous delivery features to consistently test and build code and ship it to any target.

Could be created using two ways:

1. Classic User Interface
2. YAML syntax

Support several programming languages like Python, Java, Javascript, PHP, Ruby, C#, C++ and Go

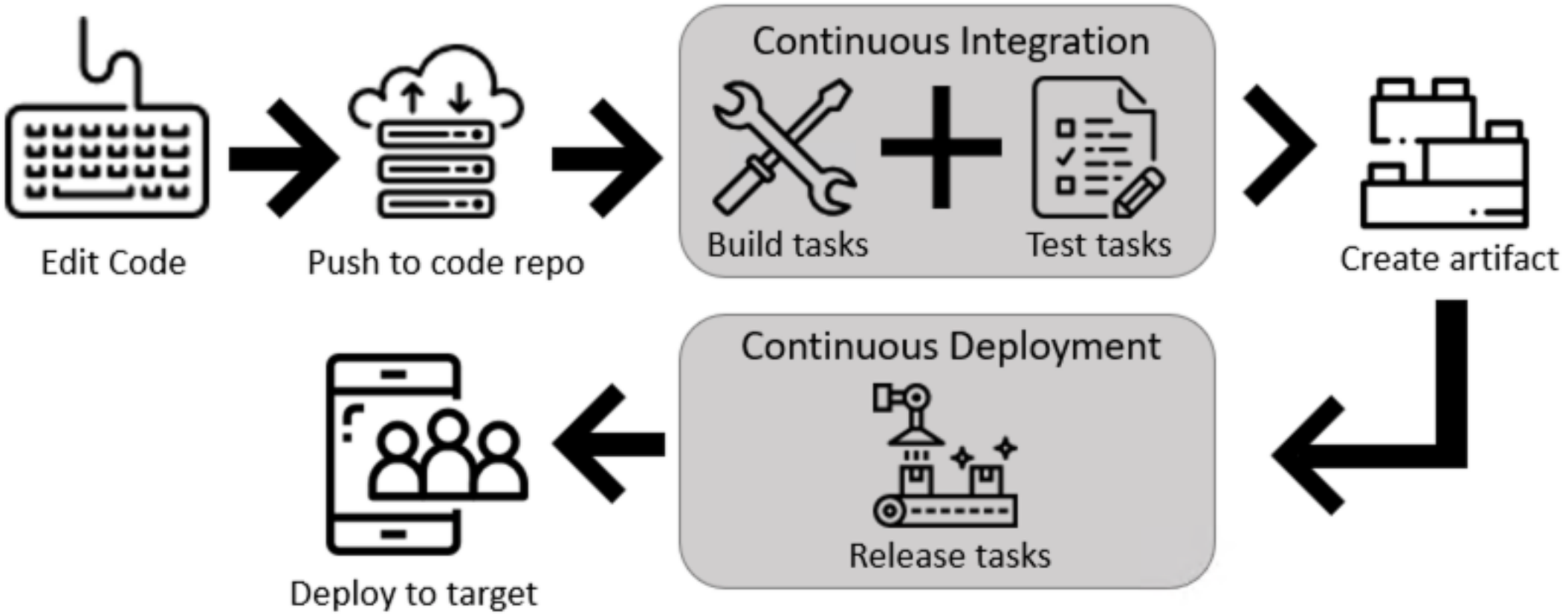
Supports Microsoft-hosted and self-hosted pipeline agents including windows, Linux, and macOS

Supports integrations with many version control systems like GitHub, Azure Repos, Bitbucket, and Subversion.

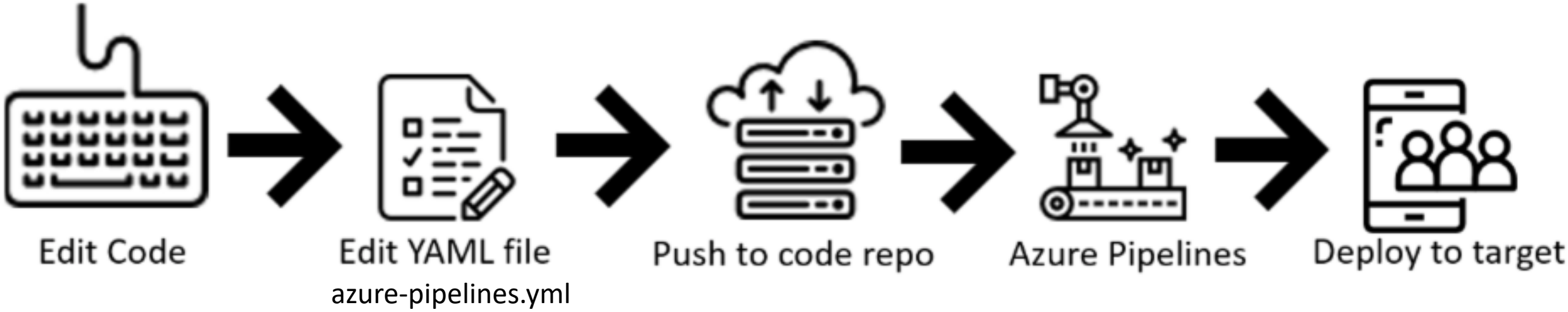
Supports deployment to multiple targets including Azure, on-premises and so on.

Supports NuGet, npm, and Maven package formats

Azure Pipelines – Classic Model



Azure Pipelines – YAML



Understanding YAML

YAML Ain’t Markup Language

Yaml.org

Human readable data serialization standard

Uses positioning and simple characters as delimiters

Structure defined by indentation

Name key value pairs separated by colon

Strings don’t require quotes

List items indicated with hyphens

https://yaml.org

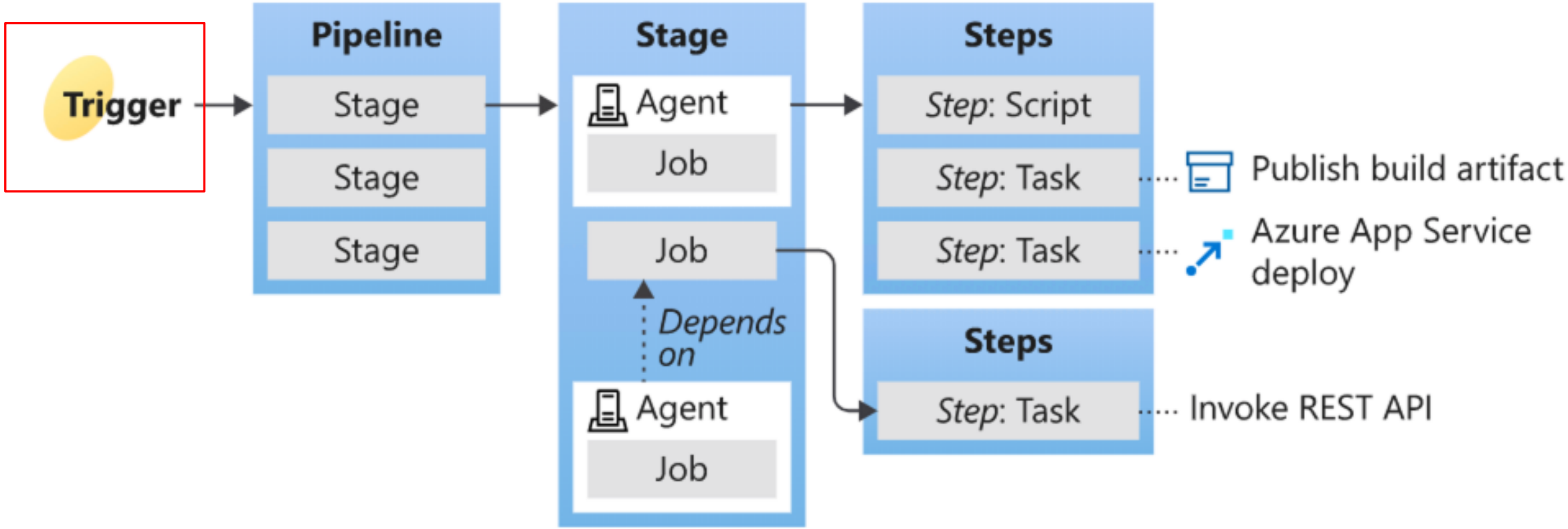
```
%YAML 1.2
---
YAML: YAML Ain't Markup Language

What It Is: YAML is a human friendly data serialization
  standard for all programming languages.

YAML Resources:
  YAML 1.2 (3rd Edition): http://yaml.org/spec/1.2/spec.html
  YAML 1.1 (2nd Edition): http://yaml.org/spec/1.1/
  YAML 1.0 (1st Edition): http://yaml.org/spec/1.0/
  YAML Issues Page:       https://github.com/yaml/yaml/issues
  YAML Mailing List:      yaml-core@lists.sourceforge.net
  YAML IRC Channel:       "#yaml on irc.freenode.net"
  YAML Reference Parser:  http://ben-kiki.org/ypaste/
  YAML Spec:              https://github.com/yaml/yaml-spec
  YAML Test Suite:        https://github.com/yaml/yaml-test-suite
  YAML Test Matrix:       https://matrix.yaml.io/
  YAML Docker Runtimes:   https://github.com/yaml/yaml-runtimes
  YAML Cookbook (Ruby):   YAML_for_ruby.html

Projects:
  C/C++:
  - libfyaml            # "C" YAML 1.2 processor                                    | YTS
  - libyaml             # "C" Fast YAML 1.1                                         | YTS
  - libcyaml            # YAML de/serialization of C data structures (using libyaml)
  - yaml-cpp            # C++ YAML 1.2 implementation
  Crystal:
  - YAML                # YAML 1.1 from the standard library
  C#/.NET:
  - YamlDotNet          # YAML 1.1/(1.2) library with serialization support          | YTS
  - yaml-net            # YAML 1.1 library
  D:
  - D-YAML              # YAML 1.1 de/serialization library with official community support | YTS
  Dart:
  - yaml                # YAML package for Dart
  Delphi:
  - Neslib.Yaml         # YAML 1.1 Delphi binding to libyaml                         | YTS
  Golang:
  - Go-yaml             # YAML support for the Go language.
  - Go-gypsy            # Simplified YAML parser written in Go.
  - goccy/go-yaml       # YAML 1.2 implementation in pure Go.
  Haskell:
  - HsYAML              # YAML 1.2 implementation in pure Haskell                    | YTS
```

Azure Pipelines – YAML



Triggers

Triggers initiates an Azure pipeline to run

Following types of triggers:

1. Continuous Integration triggers
2. Pull Request Triggers
3. Scheduled Triggers
4. Pipeline Triggers

Use Trigger: none to run the pipeline manually

Continuous Integration Triggers

Continuous integration (CI) triggers cause a pipeline to run whenever an update is pushed to the specified branches or specified tags are pushed.

```yaml
trigger:
  branches:
    include:
    - master
    - releases/*
    exclude:
    - releases/old*
```

```yaml
trigger:
  branches:
    include:
      - refs/tags/{tagname}
    exclude:
      - refs/tags/{othertagname}
```

```yaml
trigger:
  branches:
    include:
    - '*'
```

Continuous Integration Triggers

Continuous integration (CI) triggers cause a pipeline to run whenever an update is pushed to the specified branches or specified tags are pushed.

```yaml
trigger:
  batch: true
  branches:
    include:
    - master
```

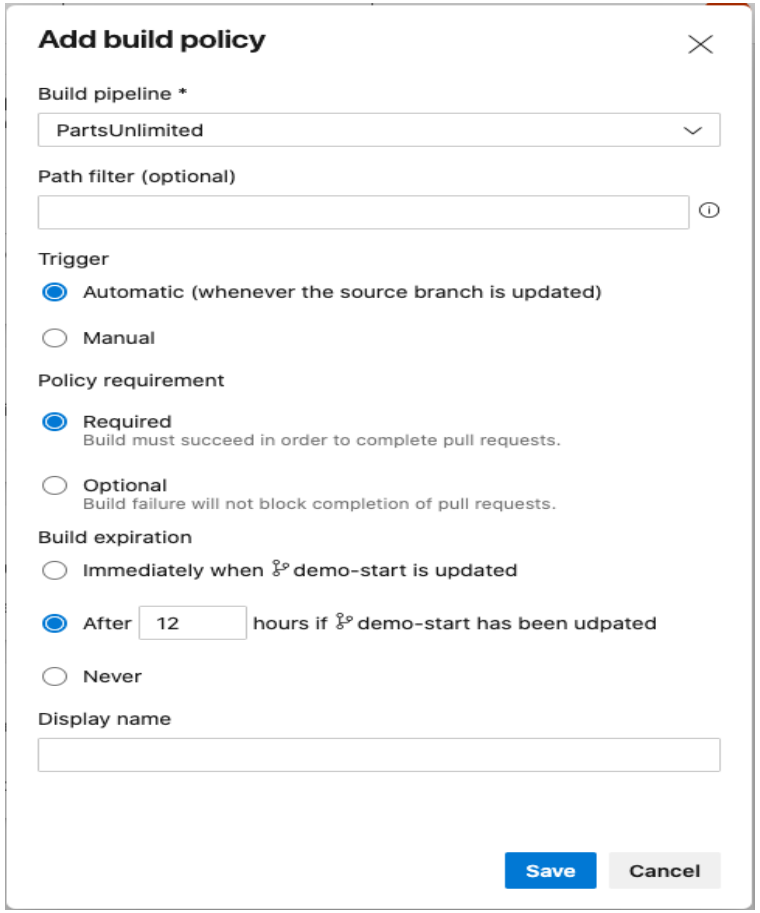
```yaml
trigger:
  branches:
    include:
    - master
    - releases/*
  paths:
    include:
    - docs/*
    exclude:
    - docs/README.md
```

```yaml
trigger:
  tags:
    include:
    - v2.*
    exclude:
    - v2.0
```

Pull Request Triggers

Pull request (PR) triggers cause a build to run whenever a pull request is opened with one of the specified target branches, or when changes are pushed to such a pull request.

Pull request triggers can be configured in branch policies.



Triggers

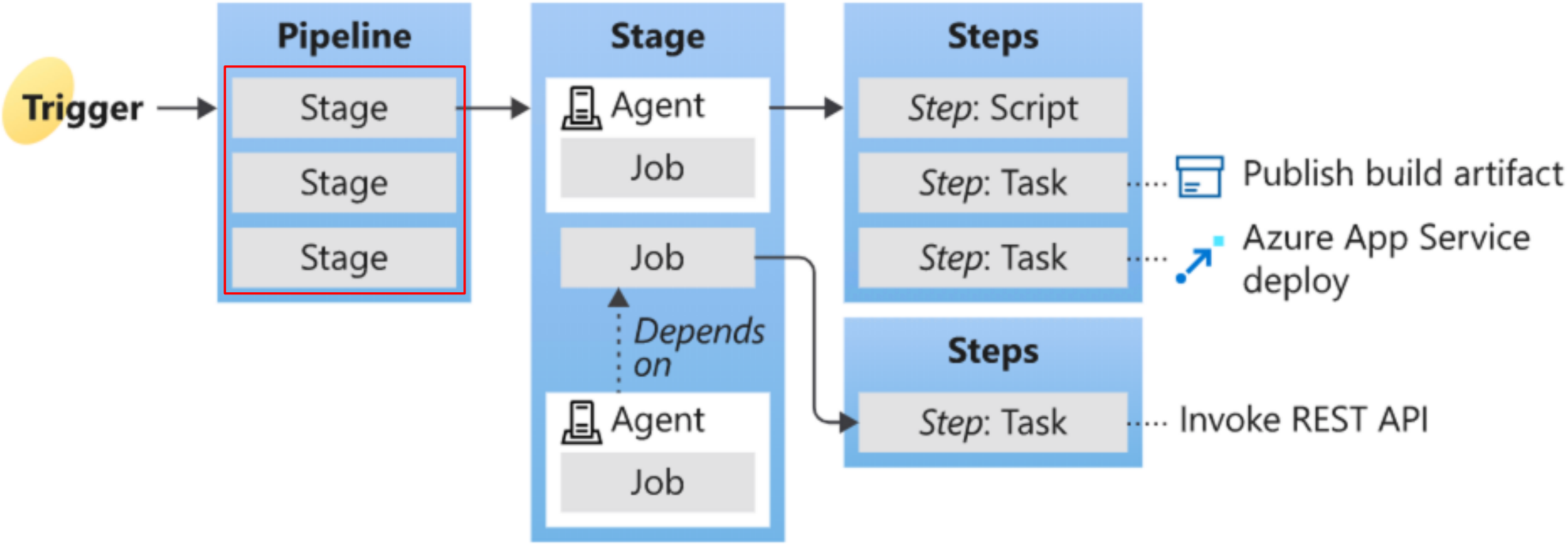
Scheduled Trigger

```yaml
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - master
    - releases/*
    exclude:
    - releases/ancient/*
- cron: "0 12 * * 0"
  displayName: Weekly Sunday build
  branches:
    include:
    - releases/*
  always: true
```

Pipeline Trigger

```yaml
resources:
  pipelines:
  - pipeline: securitylib
    source: security-lib-ci
    trigger:
      branches:
        include:
        - releases/*
        exclude:
        - releases/old*
```

Azure Pipelines – YAML



Stages

A pipeline is made up of one or more stages.

A stage is a way of organizing jobs in a pipeline and each stage can have one or more jobs

```
stages:
- stage: string  # name of the stage, A-Z, a-z, 0-9, and underscore
  displayName: string  # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  pool: string | pool
  variables: { string: string } | [ variable | variableReference ]
  jobs: [ job | templateReference]
```

Stages

Dependencies

```yaml
stages:
- stage: Build

- stage: Test
  dependsOn: Build

- stage: DeployUS
  dependsOn: Test

- stage: DeployEurope
  dependsOn:
  - Test
  - DeployUS
```

Conditions

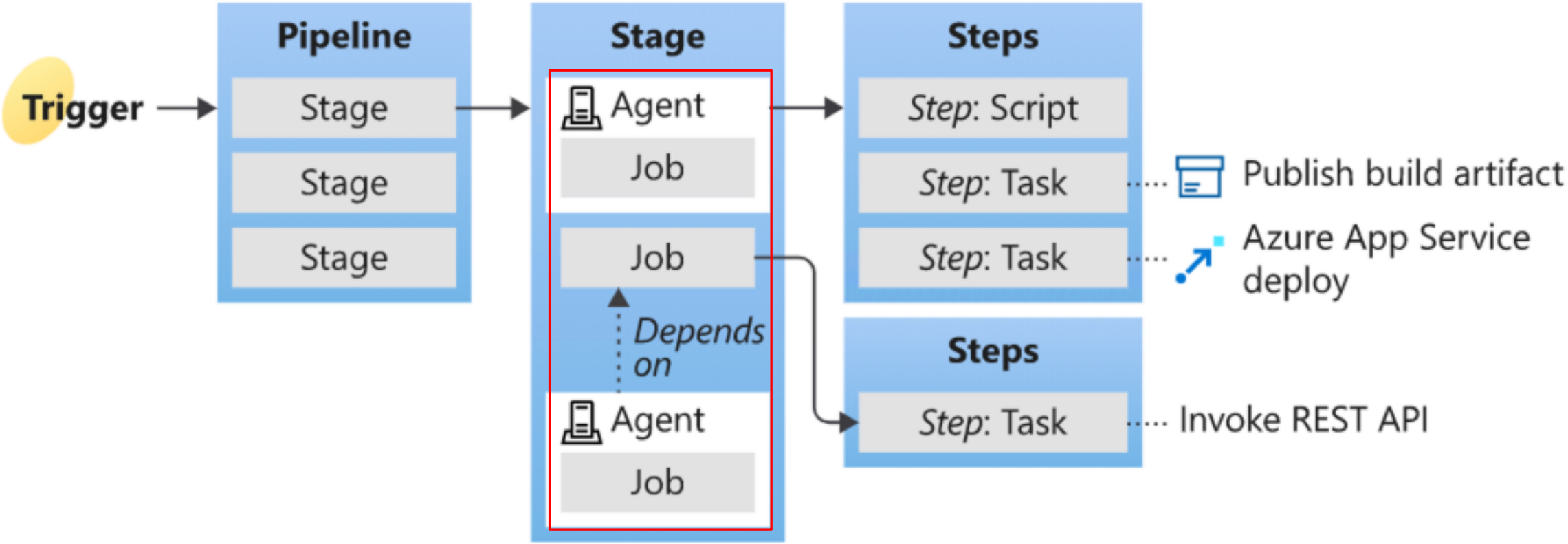
```yaml
stages:
- stage: Build

- stage: Test
  condition: succeeded()

- stage: DeployUS
  condition: succeeded(Test)

- stage: DeployEurope
  dependsOn:
  - Build
  - Test
  - DeployUS
  condition: failed(DeployUS)
```

Azure Pipelines – YAML



Jobs

A job in Azure pipelines is a collection of one or more steps.

Every pipeline has at least one job.

A job is the smallest unit of work that can be scheduled to run.

Types of jobs:

1. Agent pool jobs
2. Server Jobs
3. Container Jobs

```
jobs:
- job: string  # name of the job (A-Z, a-z, 0-9, and underscore)
  displayName: string  # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  strategy:
    parallel: # parallel strategy; see the following "Parallel" topic
    matrix: # matrix strategy; see the following "Matrix" topic
    maxParallel: number # maximum number of matrix jobs to run simultaneously
  continueOnError: boolean  # 'true' if future jobs should run even if this job fails; defaults to 'false'
  pool: pool # see the following "Pool" schema
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
  container: containerReference # container to run this job inside of
  timeoutInMinutes: number # how long to run the job before automatically cancelling
  cancelTimeoutInMinutes: number # how much time to give 'run always even if cancelled tasks' before killing them
  variables: # several syntaxes, see specific section
  steps: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
  services: { string: string | container } # container resources to run as a service container
```

Agents

Each job inside an azure pipeline is executed on an agent.

An agent is an installable software that runs one job at a time.

There are two types of agents -

Self-hosted Agents – Owned and managed by you. Gives you more control to install dependent packages for build and deployments.

Microsoft hosted Agents - Microsoft provides a fresh virtual machine every time a job needs to run. Maintenance and upgrades on the VM is managed by Microsoft. Azure pipelines provides a pre-defined agent pool (named, Azure Pipelines) with Microsoft hosted agents.

Supports Windows, Linux and MacOS.

Jobs: Dependencies

You can define the dependency of one job onto another using “depends on”.

You can execute jobs in sequence:

```yaml
jobs:
- job: Debug
  steps:
  - script: echo hello from the Debug build
- job: Release
  dependsOn: Debug
  steps:
  - script: echo hello from the Release build
```

OR, you can execute jobs in parallel:

```yaml
jobs:
- job: Windows
  pool:
    vmImage: 'vs2017-win2016'
  steps:
  - script: echo hello from Windows
- job: macOS
  pool:
    vmImage: 'macOS-10.14'
  steps:
  - script: echo hello from macOS
- job: Linux
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
  - script: echo hello from Linux
```

Jobs: Dependencies

You can define the dependency of one job onto another using “depends on”.

Fan Out:

```yaml
jobs:
- job: InitialJob
  steps:
  - script: echo hello from initial job
- job: SubsequentA
  dependsOn: InitialJob
  steps:
  - script: echo hello from subsequent A
- job: SubsequentB
  dependsOn: InitialJob
  steps:
  - script: echo hello from subsequent B
```

OR, Fan In:

```yaml
jobs:
- job: InitialA
  steps:
  - script: echo hello from initial A
- job: InitialB
  steps:
  - script: echo hello from initial B
- job: Subsequent
  dependsOn:
  - InitialA
  - InitialB
  steps:
  - script: echo hello from subsequent
```

Jobs: Conditions

Specify the conditions under which each job runs -

```yaml
jobs:
- job: A
  steps:
  - script: exit 1

- job: B
  dependsOn: A
  condition: failed()
  steps:
  - script: echo this will run when A fails

- job: C
  dependsOn:
  - A
  - B
  condition: succeeded('B')
  steps:
  - script: echo this will run when B runs and succeeds
```

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/expressions?view=azure-devops#job-status-functions

Jobs: Conditions

Specify the conditions under which each job runs -

```yaml
jobs:
- job: A
  steps:
  - script: echo hello

- job: B
  dependsOn: A
  condition: and(succeeded(), eq(variables['build.sourceBranch'], 'refs/heads/master'))
  steps:
  - script: echo this only runs for master
```

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/expressions?view=azure-devops#functions

Jobs: Execution Strategy

The matrix strategy enables a job to be dispatched multiple times, with different variable sets.

The maxParallel tag restricts the amount of parallelism.

```yaml
jobs:
- job: Test
  strategy:
    maxParallel: 2
    matrix:
      US_IE:
        Location: US
        Browser: IE
      US_Chrome:
        Location: US
        Browser: Chrome
      Europe_Chrome:
        Location: Europe
        Browser: Chrome
```

The parallel strategy enables a job to be duplicated many times.

Variables System.JobPositionInPhase and System.TotalJobsInPhase are added to each job.

```yaml
jobs:
- job: Test
  strategy:
    parallel: 5
```

Deployment Jobs

Azure pipelines can support multiple environments

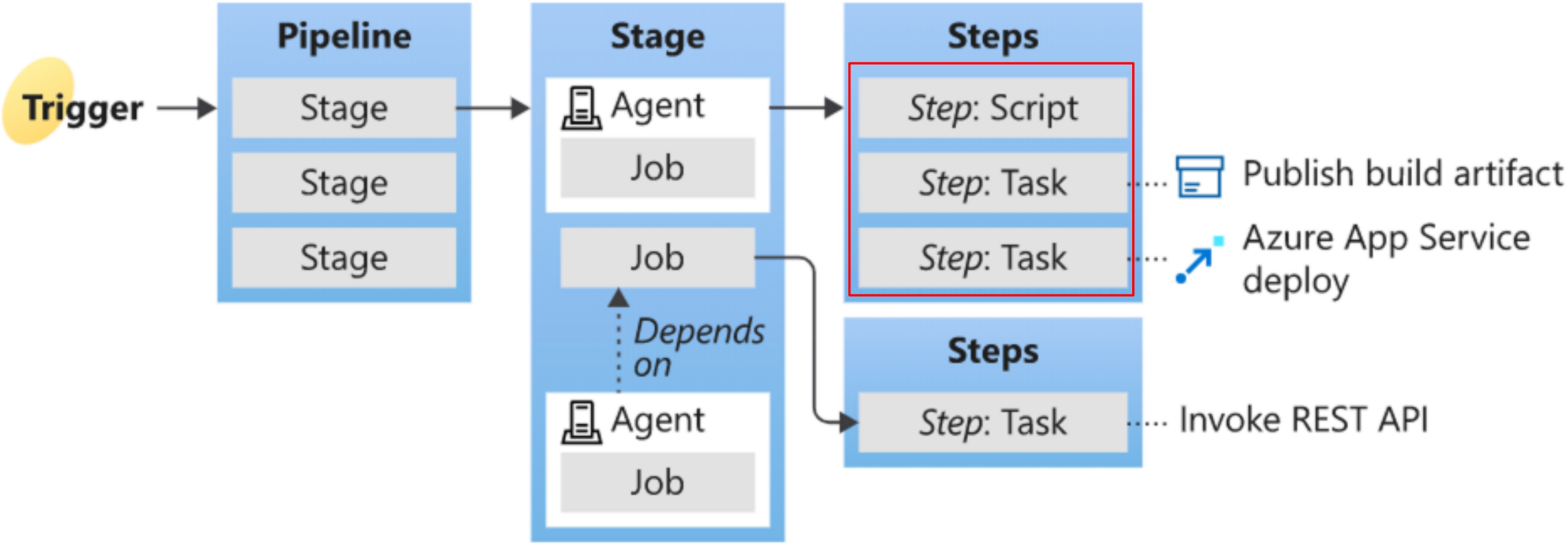
Use Deployment jobs to run steps sequentially against the environment.

Benefits –

1. Deployment jobs maintain deployment history
2. You can define the deployment strategy for your application. Supported strategies are canary, rollover and run-once.

```
jobs:
- deployment: string   # name of the deployment job (A-Z, a-z, 0-9, and underscore)
  displayName: string  # friendly name to display in the UI
  pool:                # see the following "Pool" schema
    name: string
    demands: string | [ string ]
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
  dependsOn: string
  condition: string
  continueOnError: boolean                # 'true' if future jobs should run even if this job fails; defaults to 'false'
  container: containerReference # container to run this job inside
  services: { string: string | container } # container resources to run as a service container
  timeoutInMinutes: nonEmptyString        # how long to run the job before automatically cancelling
  cancelTimeoutInMinutes: nonEmptyString  # how much time to give 'run always even if cancelled tasks' before killing them
  variables: # several syntaxes, see specific section
  environment: string  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
  strategy:
    runOnce:    #rolling, canary are the other strategies that are supported
      deploy:
        steps:
        - script: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
```

Azure Pipelines – YAML



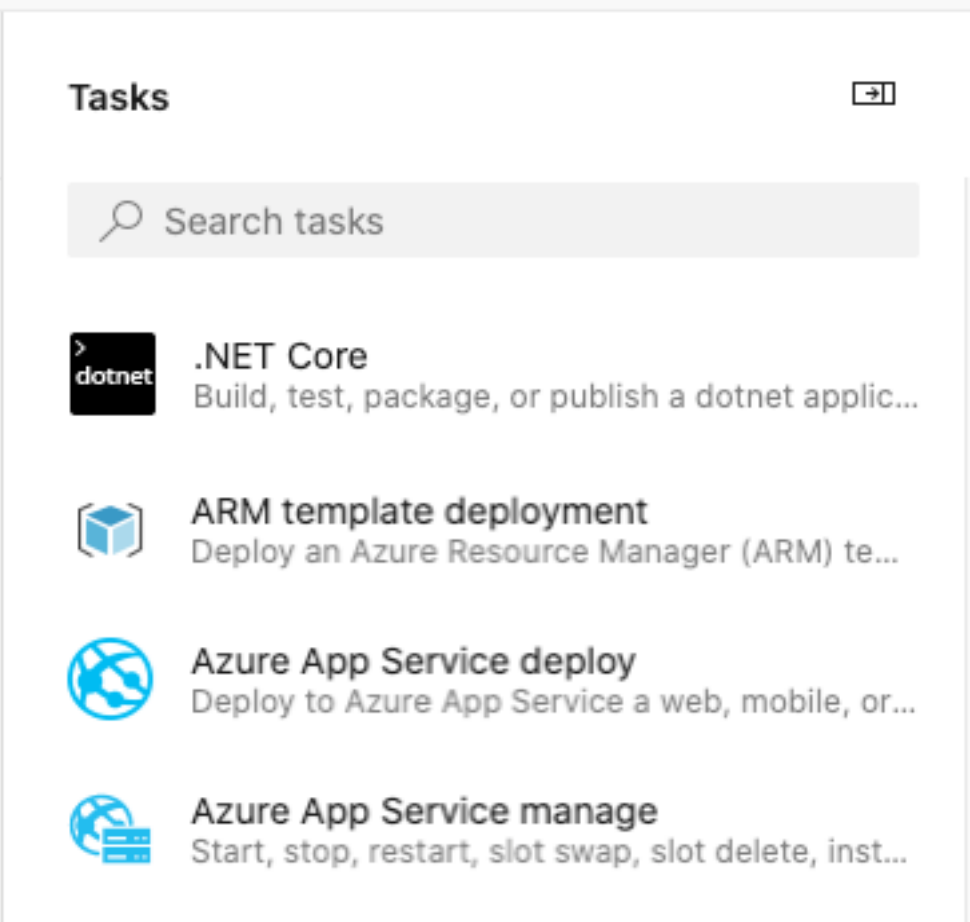
Tasks

A task is the actual execution unit within a pipeline

A job can have one or more tasks

```
steps:
- task: string  # reference to a task and version, e.g. "VSBuild@1"
  displayName: string  # friendly name displayed in the UI
  name: string  # identifier for this step (A-Z, a-z, 0-9, and underscore)
  condition: string
  continueOnError: boolean  # 'true' if future steps should run even if this step fails; defaults to 'false'
  enabled: boolean  # whether to run this step; defaults to 'true'
  target:
    container: string # where this step will run; values are the container name or the word 'host'
    commands: enum  # whether to process all logging commands from this step; values are `any` (default) or `restricted`
  timeoutInMinutes: number
  inputs: { string: string }  # task-specific inputs
  env: { string: string }  # list of environment variables to add
```

Built-in Tasks



Templates

Templates let you define reusable content, logic, and parameters

Templates can be used two ways –

1. To insert reusable content
2. To control what is allowed in a pipeline

```yaml
# File: simple-param.yml
parameters:
- name: yesNo # name of the parameter; required
  type: boolean # data type of the parameter; required
  default: false

steps:
- script: echo ${{ parameters.yesNo }}
```

```yaml
# File: azure-pipelines.yml
trigger:
- master

extends:
  template: simple-param.yml
  parameters:
    yesNo: false # set to a non-boolean value to have the build fail
```

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/templates

Demo

Classic vs YAML pipelines

Certain features are either supported by classic or YAML pipelines and not the other

For example –

Container jobs are supported in YAML but not in classic

Task Groups are supported in classic but not in YAML

Templates are supported only in YAML and not in classic

Version control in YAML is the primary benefit

https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops#feature-availability

Resources for further reading

1. The official YAML Website - https://yaml.org/
2. Azure Pipelines: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines
3. Azure Pipelines YAML Schema reference - https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema
4. Azure DevOps Demo Generator: https://azuredevopsdemogenerator.azurewebsites.net/


thank
you