YAML

Yet Another Markup Language

.NET

YAML is a human-readable data serialization standard for all programming languages.
It's a strict superset of JSON, with the addition of syntactically significant newlines and indentation.

HTTPS://YAML.ORG/

What is YAML?

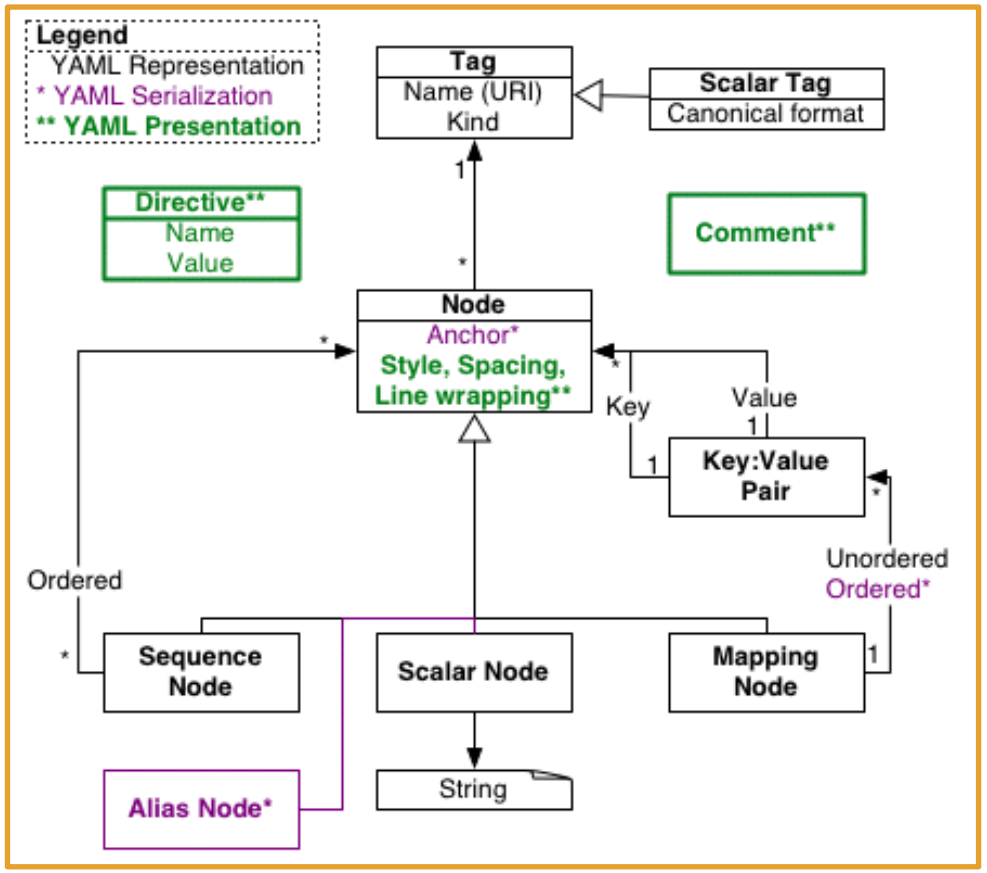
https://yaml.org/spec/1.2/spec.html
http://www.yamllint.com/

YAML is a Unicode-based data serialization language. It's designed around the common native data types of Agile programming languages.

YAML is useful for configuration, messaging, object persistence, and data auditing.

The design goals for *YAML*, in decreasing priority, are:

- Be easily readable by humans.
- Be portable between programming languages.
- Match the native data structures of Agile languages.
- Have a consistent model to support generic tools.
- Support one-pass processing.
- Be expressive and extensible.
- Be easy to implement and use.



Azure Pipelines and YAML

https://devblogs.microsoft.com/devops/announcing-general-availability-of-azure-pipelines-yaml-cd/
https://azure.microsoft.com/en-us/services/devops/logout/?nav=min

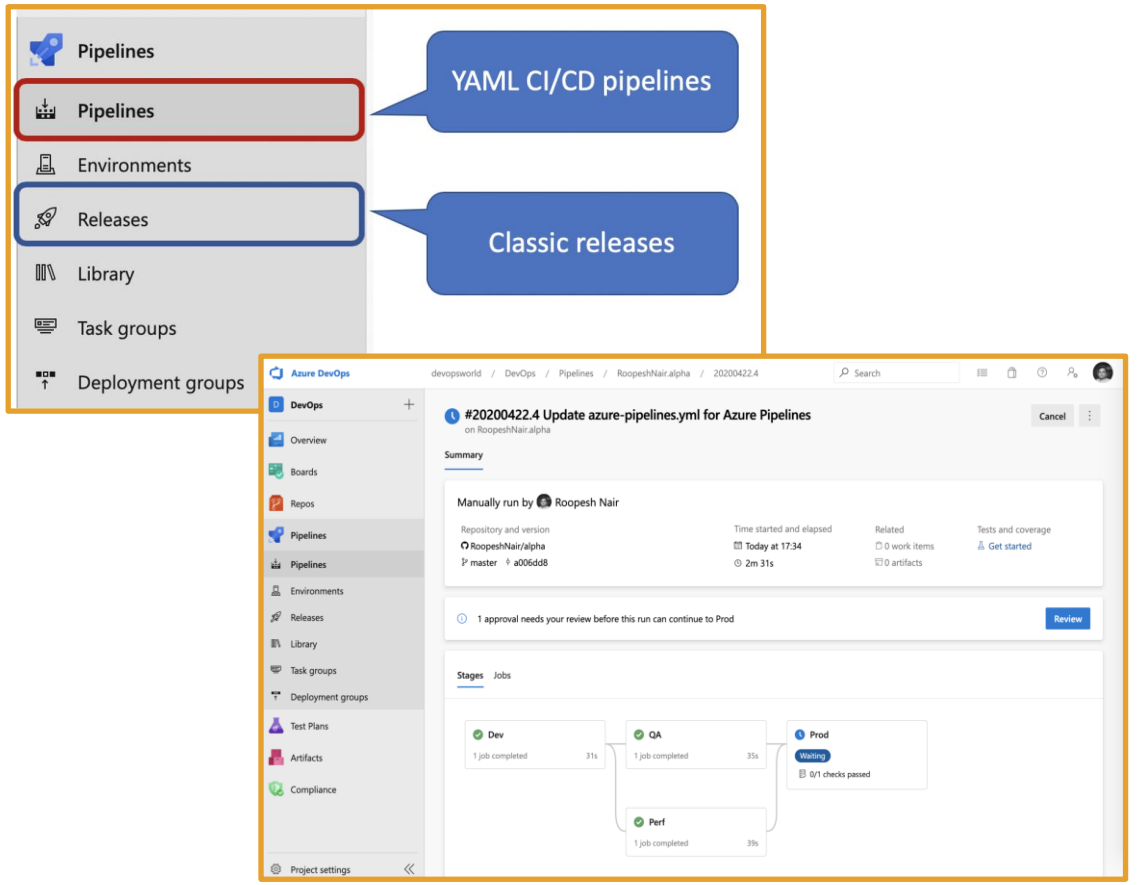
---

Microsoft's ***Azure Pipelines*** offers a way to use .yml files to configure CI/CD Pipelines.

A ***pipeline*** is defined using a YAML file. This .yml file is placed at the root of your project when you set up your Azure Pipeline.

The YAML file controls the build, test, and deploy stages of your application. It is started by a ***trigger*** activated when certain conditions are met in your repository.

The ***trigger*** knows about a push, PR, etc, to the repository because of a ***Web Hook*** on the repository.



YAML Triggers

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cyaml-example#push-trigger
https://docs.microsoft.com/en-us/azure/devops/pipelines/build/triggers?tabs=yaml&view=azure-devops#pr-triggers
https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/github?view=azure-devops&tabs=yaml#ci-triggers

Triggers are how you automatically build your application. They are placed at the top of your YAML file. Azure Pipelines watches for your designated ***trigger*** and will automatically start running your YAML instructions when the ***trigger*** event is detected.

```yaml
name: '$(date:yyyyMMdd)$(rev:rr)'
#what branch we are watching
trigger:
  - 'master'
pr: 'none'
pool:
  vmImage: 'ubuntu-latest'
variables:
  instructor: 'Mark'
  sdkVersion: '3.1.x'
  solutionPath: 'pipelineMvcDemo/pipelineMvcDemo.csproj'
  buildConfiguration: 'Release'
```

A ***push trigger*** specifies that a 'git push' to a particular branch will cause a build to run. If you specify a "no push" trigger, pushes to any branch trigger a build.

```yaml
trigger:
- master
- develop
```

```yaml
trigger: none # will disable CI builds (but not PR builds)
```

A ***Pull Request Trigger*** will start a build when a PR is made to the specified branch. You can also specify 'none', which will disable PR triggers.

```yaml
pr:
- master
- develop
```

```yaml
pr: none # will disable PR builds
```

If you don't specify pr: none, all pr creation's will trigger a build!!

YAML and Pipeline Structure

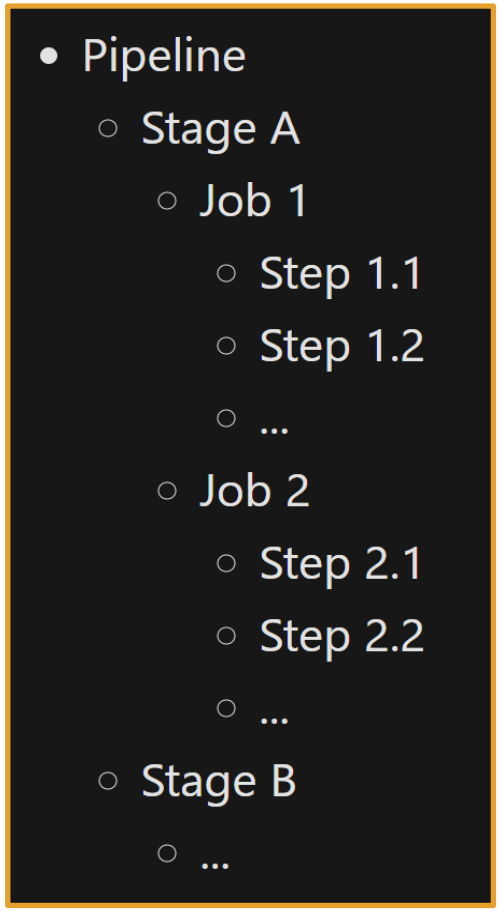
https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema
https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops

Azure Pipelines supports CI/CD using a .yml file. The .yml file (YAML) is written in ***YAML*** Syntax and contains instructions that ***Azure Pipelines*** uses to build, test, report, publish, and deploy an application.

A ***pipeline*** is made up of one or more "stages" that describe processes.

- Stages are the major divisions in the deployment process (building, testing and deployment).
- Each Stage is divided into Jobs.
- A Job is a unit of work assignable to one machine and is divided into Steps
- Each Step is a series of Tasks, scripts, or references to external templates.
- Tasks are the smallest units of work (executable operation) in the pipeline.

Simple ***pipelines*** can omit multiple stages and jobs as needed. ***Azure Pipelines*** does not support all ***YAML*** features.



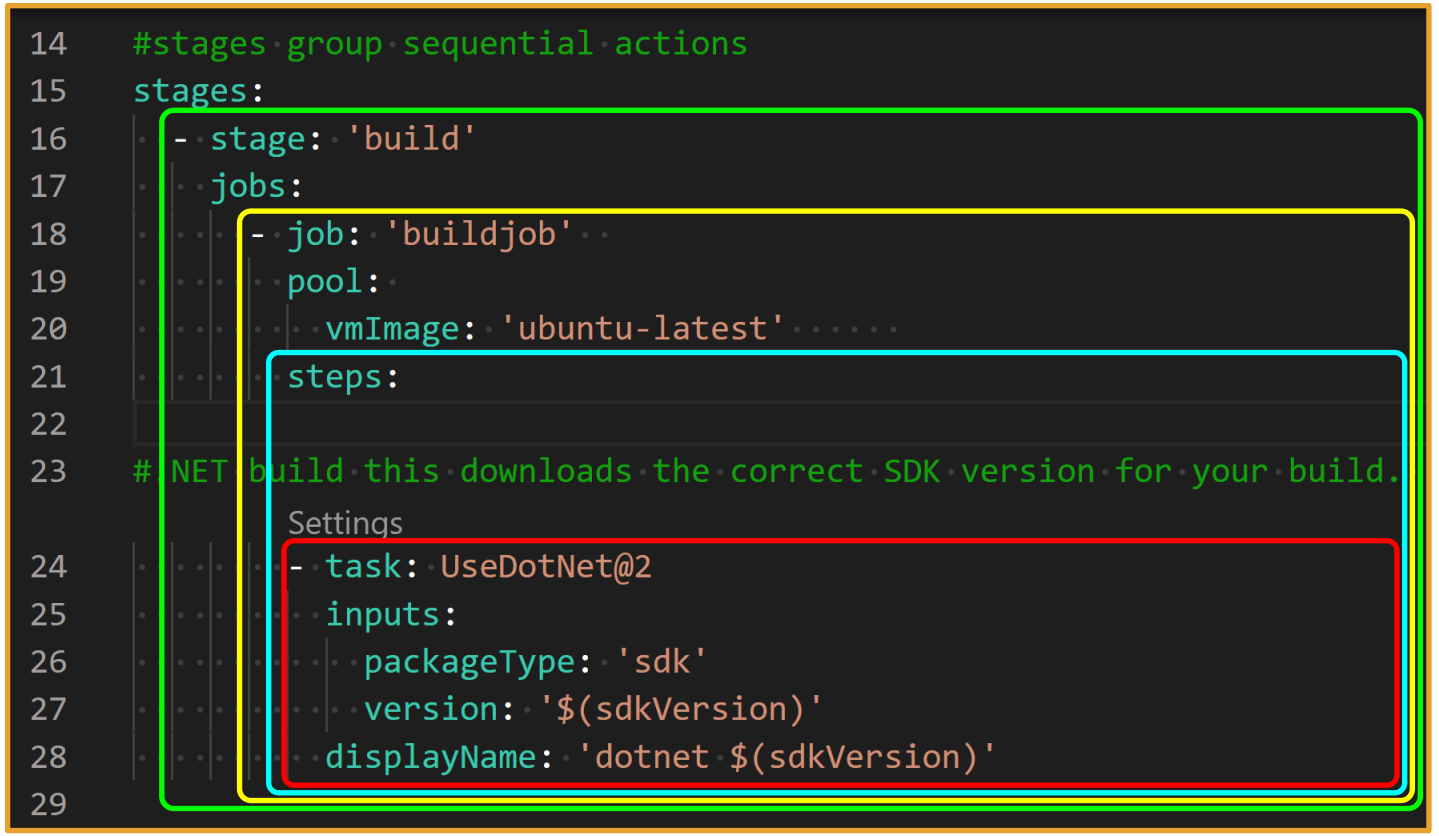
YAML and Pipeline Structure

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema
https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops

---

A ***pipeline*** is made up of one or more “stages” that describe processes.

- Stages are the major divisions in the deployment build, test and deploy process.
- Each Stage is divided into Jobs.
- A Job is a unit of work assignable to one machine.
- Each job is divided into steps
- Each Step is divided into Tasks. The Task is the smallest unit of work in the pipeline.
- If you have more than one Stage to list, they can be listed inside a “Stages” section. The same is for Jobs.



YAML Stage

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#stage

A ***Stage*** is a collection of related ***Jobs***. By default, ***Stages*** run sequentially.

In this example:

- A region of ***Stages*** is declared then given a name.
- Each stage has a ***Jobs*** region where one or more ***Jobs*** are listed and named.
- Each ***Job*** has a ***Steps*** region where one or more ***Tasks*** can be listed.
- A script is a type of ***Task*** and represents what you could type in the command line.

```
stages:
- stage: Build
  jobs:
  - job: BuildJob
    steps:
    - script: echo Building!
- stage: Test
  jobs:
  - job: TestOnWindows
    steps:
    - script: echo Testing on Windows!
  - job: TestOnLinux
    steps:
    - script: echo Testing on Linux!
- stage: Deploy
  jobs:
  - job: Deploy
    steps:
    - script: echo Deploying the code!
```

```
stages:
- stage: BuildWin
  displayName: Build for Windows
- stage: BuildMac
  displayName: Build for Mac
  dependsOn: [] # by specifying a
```

This example shows how to run two stages in parallel (async). Specify an empty array with 'dependsOn:' to run a stage without waiting for the preceding stage to complete.

YAML Job

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#job

---

A ***Job*** is where you can add a reference to a container.

```
jobs:
- job: RunsInContainer
  container: ubuntu:16.04
```

```
# Docker Hub image reference
```

A ***Job*** is a collection of ***Steps*** run by an agent or on a server. ***Jobs*** can run conditionally and might depend on earlier ***Jobs***.

```
jobs:
- job: MyJob
  displayName: My First Job
  continueOnError: true
  workspace:
    clean: outputs
  steps:
  - script: echo My first job
```

A ***deployment job*** is a special type of ***job***. It is a collection of ***steps*** to run sequentially against the environment.

```
jobs:
  # track deployments on the environment
- deployment: DeployWeb
  displayName: deploy Web App
  pool:
    vmImage: 'Ubuntu-16.04'
  # creates an environment if it doesn't exist
  environment: 'smarthotel-dev'
  strategy:
    # default deployment strategy, more coming...
    runOnce:
      deploy:
        steps:
        - script: echo my first deployment
```

YAML Steps

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#steps

---

A ***job*** is made up of one or more ***steps***. Each ***step*** runs in its own process and has access to the pipeline workspace. Environment variables aren't preserved between ***steps***, but file system changes are. Supported ***Tasks*** in Azure Pipelines are script, bash, pwsh, PowerShell, Checkout, task, and step templates.

```yaml
steps:
- script: echo This runs in the default shell on any machine
- bash: |
    echo This multiline script always runs in Bash.
    echo Even on Windows machines!
- pwsh: |
    Write-Host "This multiline script always runs in PowerShell Core."
    Write-Host "Even on non-Windows machines!"
```

YAML Variables

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#variables

You can add hard-coded values directly or reference a variable group.

You can specify variables at the pipeline, stage, or job level by using the 'variables:' keyword followed by the name and value of the variable in single quotes.

```yaml
variables:      # pipeline-level
  MY_VAR: 'my value'
  ANOTHER_VAR: 'another value'

stages:
- stage: Build
  variables:    # stage-level
    STAGE_VAR: 'that happened'

  jobs:
  - job: FirstJob
    variables:  # job-level
      JOB_VAR: 'a job var'
    steps:
    - script: echo $(MY_VAR) $(STAGE_VAR) $(JOB_VAR)
```

Template References

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#template-references

You can import reusable sections of your pipeline from a separate (***template***) file. These separate files are known as ***templates***. ***Templates*** can include other ***templates***.

Azure Pipelines supports four kinds of ***templates:***

- Stage
- Job
- Step
- Variable

Template that accepts parameters

```yaml
# File: stages/test.yml

parameters:
  name: ''
  testFile: ''

stages:
- stage: Test_${{ parameters.name }}
  jobs:
  - job: ${{ parameters.name }}_Windows
    pool:
      vmImage: vs2017-win2016
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
  - job: ${{ parameters.name }}_Mac
    pool:
      vmImage: macos-10.14
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
```

.yml file which calls the template using parameters

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

stages:
- template: stages/test.yml  # Template reference
  parameters:
    name: Mini
    testFile: tests/miniSuite.js

- template: stages/test.yml  # Template reference
  parameters:
    name: Full
    testFile: tests/fullSuite.js
```

Template References

https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#template-references

In this example:

- The upper image shows the template .yml file.
- The lower image is the .yml file that called the template with arguments.
- The lower YAML file calls the template two times.
- The Template instantiates an object at the top with the keyword 'parameters' to accept arguments passed in.
- It then references the values of the object while creating jobs and using npm commands.

Template that accepts parameters

```yaml
# File: stages/test.yml

parameters:
  name: ''
  testFile: ''

stages:
- stage: Test_${{ parameters.name }}
  jobs:
  - job: ${{ parameters.name }}_Windows
    pool:
      vmImage: vs2017-win2016
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
  - job: ${{ parameters.name }}_Mac
    pool:
      vmImage: macos-10.14
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
```

.yml file which calls the template using parameters

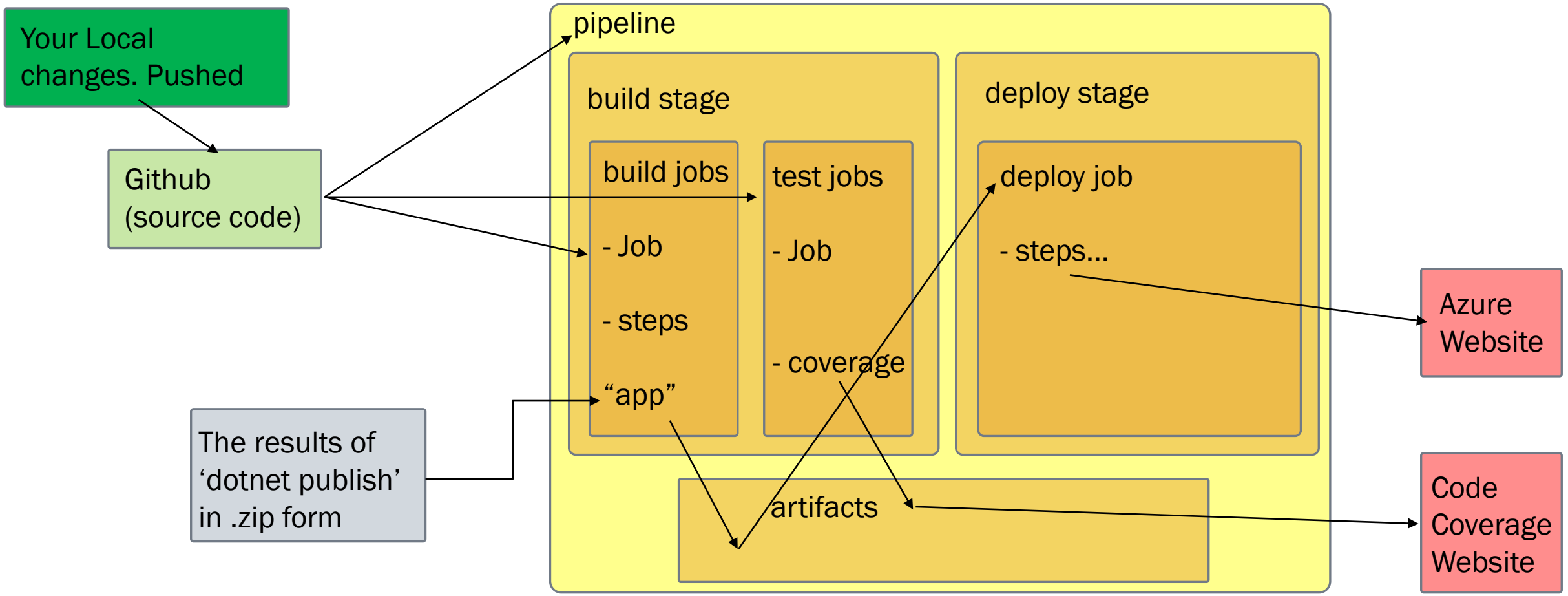
```yaml
# File: azure-pipelines.yml

stages:
- template: stages/test.yml  # Template reference
  parameters:
    name: Mini
    testFile: tests/miniSuite.js

- template: stages/test.yml  # Template reference
  parameters:
    name: Full
    testFile: tests/fullSuite.js
```

Convert these .json files to .yml

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

```json
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Pipeline Deployment Workflow



Additional Resources

- Build your first Azure Pipeline
  - https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/dotnet-core?view=azure-devops
- See code coverage in Azure Pipeline
  - https://docs.microsoft.com/en-us/azure/devops/pipelines/test/review-code-coverage-results?view=azure-devops
- Deploy to Azure App Service using Visual Studio Code
  - https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/deploy-to-azure-vscode?view=azure-devops