Configuring Resources, Tasks, and Jobs in a Pipeline

Richard Seroter
VICE PRESIDENT, PIVOTAL

@rseroter

Overview

Learn about and use tasks

Implement resources

Understand and build jobs

Concourse Building Blocks

Resource	Resource Type	Task
Job	Pipeline	Build

The Role of Tasks

- Similar to a function, with input and output
- Smallest configurable unit
- Runs in containers using designated image
- Tasks specify a statement to execute

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

- Specifies which platform the task should run on. Options include `windows`, `linux`, or `darwin`.

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

- Sets which container image to run the task in
- `repository` is a required source parameter, and optional ones like `tag` or `username`/`password` exists based on the resource type
- May use your own Docker image if you want to include pre-baked bits

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

- Specify the artifacts available in the current directory when the task runs
- Can optionally provide a `path` parameter if it differs from the input name
- Inputs may come from CLI parameters, a `get` step in the build plan, or from the `outputs` of a previous task

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

- A directory available to later steps in the build plan
- Directory is automatically created when the task is run
- Any artifacts you want available to later tasks need to be placed here by the current task

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: ubuntu}

inputs:
  name: demo-folder

outputs:
- name: compiled-app

run:
  path: find
  args: [.]
```

- This represents the command to run in the container
- The `path` attribute typically points to a script provided by a task input. It may also reference a direct command like `bash`

Looking at Task Inputs and Outputs

Resources are the only durable storage mechanism in a pipeline

Task inputs produce directories full of artifacts within the container

Task outputs are stored on the worker filesystem and mounted into containers for subsequent tasks within the same job

Demo

- Create a new task definition
- Execute a standalone task
- Augment our task with inputs and outputs
- Use external script for “run” command

The Role of Resources

Represent the external inputs and outputs of a job in a pipeline

Resources have a “type” and behavior when retrieved or published to

Concourse provides out-of-the-box resource types

```
---
resource_types:
- name: azure-blobstore
  type: docker-image
  source:
    repository: [coordinates]

resources:
- name: source-code
  type: git
  icon: github-circle
  source:
   uri: [repo address]
   branch: master
- name: artifact-repo
  type: azure-blobstore
  icon: azure
  source:
   storage_account_name: [name]
   storage_account_key: [key]
   container: [container name]
   versioned_file: [file name]
```

```
---
resource_types:
- name: azure-blobstore
  type: docker-image
  source:
    repository: [coordinates]

resources:
- name: source-code
  type: git
  icon: github-circle
  source:
   uri: [repo address]
   branch: master
- name: artifact-repo
  type: azure-blobstore
  icon: azure
  source:
   storage_account_name: [name]
   storage_account_key: [key]
   container: [container name]
   versioned_file: [file name]
```

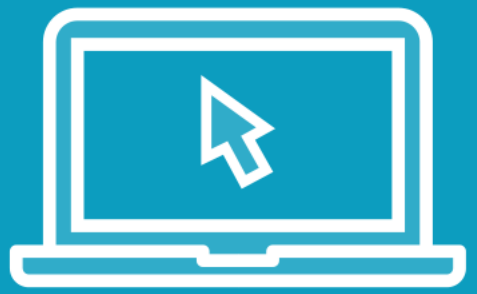
- Every resource has a “type” which determines detected version, what is retrieved, and what happens on “put”
- May also declare additional optional parameters as needed by the resource type
- Can set `check_every` to override 1m polling period
- ”Core” types don’t need to be declared in the pipeline definition

```
---
resource_types:
- name: azure-blobstore
  type: docker-image
  source:
    repository: [location]

resources:
- name: source-code
  type: git
  icon: github-circle
  source:
   uri: [repo address]
   branch: master
- name: artifact-repo
  type: azure-blobstore
  icon: azure
  source:
   storage_account_name: [name]
   storage_account_key: [key]
   container: [container name]
   versioned_file: [file name]
```

- Choose a useful `name`, as it gets referenced elsewhere in the pipeline
- The `type` maps to core or custom resource types
- The `icon` property adds a visual indicator to the resource
- A `source` is specific to the type of resource you're working with

Demo

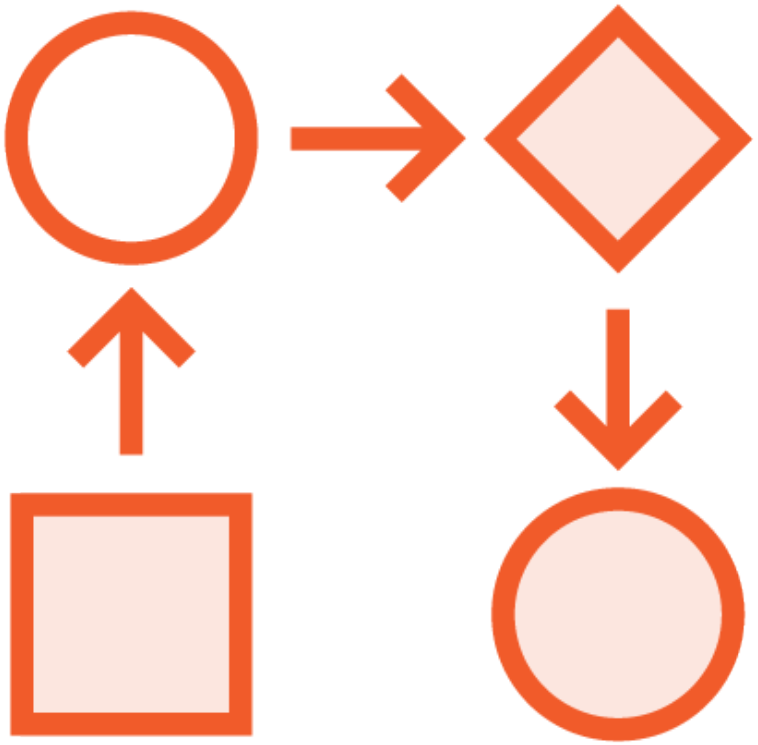


Explore the resource types available on the worker node

Create a simple pipeline that uses resources

Make resources available to our task

The Role of Jobs



Jobs are made up of build plans that set the sequence of steps to execute

A job has a single build plan

Offers three types of steps (get, put, task) to process resources

It's possible to have steps that execute on success, failure, or abort

Scope jobs to a particular outcome that results in a usable resource

```
---

...


jobs:
 - name: job-pluralsight
   plan:
   - get: source-code
   - task: first-task
     config:
       platform: linux
       image_resource:
         type: docker-image
         source: {repository: ubuntu}
       inputs:
       - name: source-code
       outputs:
       - name: compiled-app
       run:
         path: source-code/ci/script1.sh
```

```
---

...

jobs:
 - name: job-pluralsight
   plan:
   - get: source-code
   - task: first-task
     config:
       platform: linux
       image_resource:
         type: docker-image
         source: {repository: ubuntu}
       inputs:
       - name: source-code
       outputs:
       - name: compiled-app
       run:
         path: source-code/ci/script1.sh
```

- Use descriptive names for your jobs

```
---

...


jobs:
 - name: job-pluralsight
   plan:
   - get: source-code
   - task: first-task
     config:
       platform: linux
       image_resource:
         type: docker-image
         source: {repository: ubuntu}
       inputs:
       - name: source-code
       outputs:
       - name: compiled-app
       run:
         path: source-code/ci/script1.sh
```

- Declare a plan and your steps
- Define which resources you want to `get` and make available to tasks
- A `get` step can have a properties for version to retrieve, auto-trigger behavior, and which jobs must have have passed prior
- `put` steps would typically be at the end of the plan and have access to `outputs` populated by tasks

```
---

...

jobs:
 - name: job-pluralsight
   plan:
   - get: source-code
   - task: first-task
     config:
       platform: linux
       image_resource:
         type: docker-image
         source: {repository: ubuntu}
       inputs:
       - name: source-code
       outputs:
       - name: compiled-app
       run:
         path: source-code/ci/script1.sh
```

- Task steps can either have a `config` property or `file` property
- A `file` property means that the step points to another .yml file with the task configuration
- Choosing to `get` a resource does not automatically make it available to a task without explicitly declaring an `input`

Demo

Create a pipeline with a single job

Show how to pass state between steps in a job

Add a second job that puts packaged code into an AWS S3 bucket

Add a third job that retrieves the packaged code from AWS S3 and unpacks it

Summary

Learn about and use tasks

Implement resources

Understand and build jobs