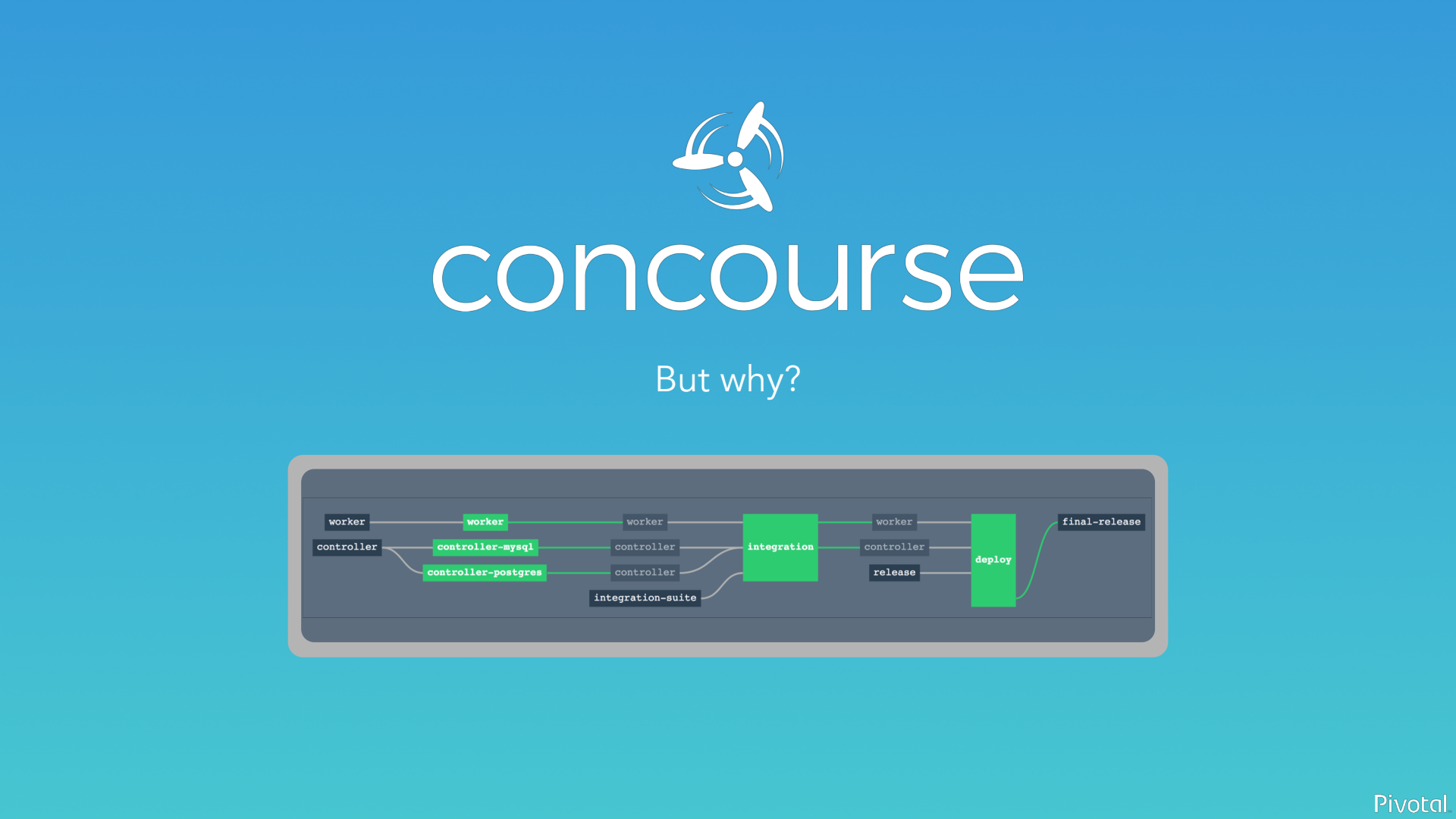

concourse
But why?
worker
worker
worker
controller
controller-mysql
controller
controller-postgres
controller
integration-suite
integration
worker
controller
release
deploy
final-release

Continuous Integration

AUTOMATION.
Integrate tools and automate processes from testing to builds and deployment

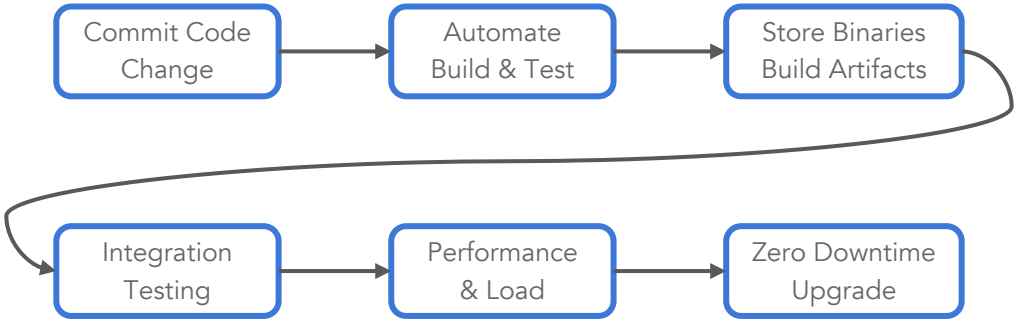
AGILITY.
Push updates on regular basis with no downtime to improve customer experience and time to market

QUALITY.
Reduce feedback loop using test-driven development to surface problems sooner and be responsive

SPEED.
Release more frequently with smaller bits will reduce complexity and improve time-to-market

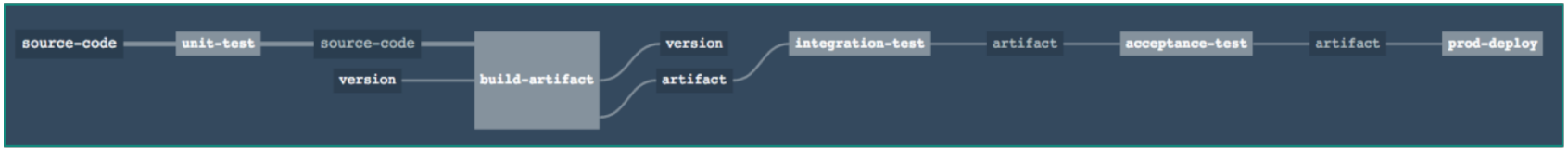
Continuous Integration

Continuous Delivery - an approach in which teams ensure that **every change** to the system is **releasable**, and that we can release any version at the **push of a button**



Continuous Integration

The Pipeline:



Concourse moves to realize the conceptual **delivery model** in visual **pipelines**!

Continuous Integration

- No support for pipelines
- Debugging hard

CLOUDFOUNDRY

WHY CONCOURSE?

- Configuration vs Run
- Pipeline hierarchy too deep

- Snowflakes
- Too many (unsupported) plugins!

Continuous Integration

Concourse Core Concepts

Task
The execution of a script in an isolated environment with dependent resources available to it.

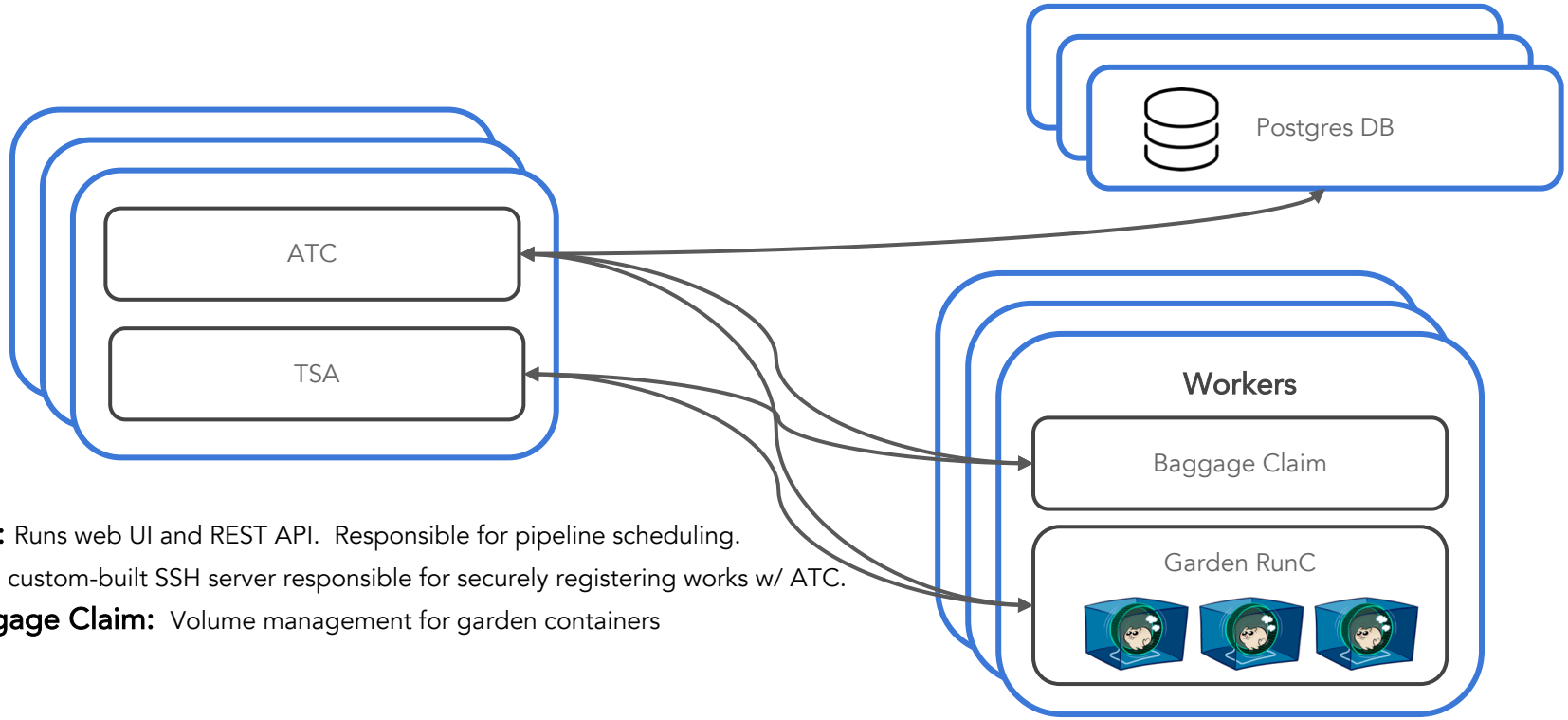
Resource
Any entity that can be checked for new versions, pulled down at a specific version, and/or pushed up to idempotently create new versions.

Job
A description of some actions to perform when dependent resources change (or when manually triggered)

Continuous Integration

Anger Optimized UI -- You shouldn't get angry at your CI because you can't find the broken build in your pipeline

Continuous Integration



ATC: Runs web UI and REST API. Responsible for pipeline scheduling.

TSA: custom-built SSH server responsible for securely registering works w/ ATC.

Baggage Claim: Volume management for garden containers

concourse

Setup Env

Fly CLI

The fly tool is a command line interface to Concourse.

It is used for all interaction with Concourse minus the viewing of actual pipelines.

We will utilize fly to uploading new pipeline configuration into a running Concourse, connect to a shell in one of your build's containers, and more!

Download the CLI, rename, and move it into your PATH:

- Linux https://github.com/concourse/concourse/releases/download/v3.5.0/fly_linux_amd64
- MacOS - https://github.com/concourse/concourse/releases/download/v3.5.0/fly_darwin_amd64
- Windows - https://github.com/concourse/concourse/releases/download/v3.5.0/fly_windows_amd64.exe

Example (MacOS):

```
wget https://github.com/concourse/concourse/releases/download/v3.5.0/fly_darwin_amd64
chmod +x fly_*
mv fly_* /usr/local/bin/fly
```

Fly CLI

Utilize the **Fly CLI** to verify connectivity to Concourse, the CI infrastructure.

Example:

```
fly --target concourse-workshop login --concourse-url https://ci.oskoss.com -k --team-name Main
logging in to team 'Main'

username: <USERNAME>
Password: <PASSWORD>

target saved
```

Setup Git

We will utilize **git** to pull the latest configuration for deploying each **concourse pipeline.**
Download and install Git CLI:

- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Git Clone (download) the BOSH Lab Repository:

- git clone https://github.com/occelebi/concourse-tutorial
- mv concourse-tutorial demo-pipelines

- cd demo-pipelines
- git clone https://github.com/Oskoss/Workshops-2.0.git

- git clone https://github.com/starkandwayne/concourse-tutorial.git

concourse

(1) Concourse Running Tasks

Running a Task

The central concept of Concourse is to run **tasks**

You can run them directly from the **command line** as below, or from within pipeline **jobs** as is done for the rest of the workshop

Example (MacOS):

```
cd demo-pipelines/01_task_hello_world/
fly --target concourse-workshop execute --config task_hello_world.yml
```

Running a Task

Each task in Concourse runs within a Docker image.

In the output below you can see Concourse is downloading a busybox Docker image.

```
Output (Part 1):
Pulling busybox@sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7...
sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7: Pulling from library/busybox
03b1be98f3f9: Pulling fs layer
03b1be98f3f9: Verifying Checksum
03b1be98f3f9: Download complete
03b1be98f3f9: Pull complete
Digest: sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7
Status: Downloaded newer image for busybox@sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7

Successfully pulled busybox@sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7.
```

Tip: It will only need to do this once as concourse rechecks every time that it has the latest busybox image.

Running a Task

Each task in Concourse runs some form of bash command.

In the output below you can see Concourse ran the echo command with ‘hello world’, and it ran successfully.

Output (Part 1):
running echo hello world
hello world
succeeded

Running a Task

Notice in the fly cli command we ran earlier we specified '--config task_hello_world.yml'

Examining this file we see the command and arguments we requested to run, along with the docker image to run the command in

```
---
platform: linux

image_resource:
  type: docker-image
  source: {repository: busybox}

run:
  path: echo
  args: [hello world]
```

Tip: The platform tag determines the pool of workers that the task can run against.

Running a Task

Try changing the ‘task_hello_world.yml’ with the following and re-running the task

```
---
platform: linux

image_resource:
  type: docker-image
  source: {repository: ubuntu, tag: "14.04"}

run:
  path: uname
  args: [-a]
```

Tip: To rerun the task: `fly --target concourse-workshop execute --config task_hello_world.yml`

Key Takeaways

Task executions can be used to more closely replicate the state in CI when weeding out flakiness, or as a shortcut for local development so that you don't have to upload every single resource from your local machine.

Selecting any base image (or image_resource when configuring a task) allows your task to have any prepared dependencies that it needs to run.

Instead of installing dependencies each time during a task you might choose to pre-bake them into an image to make your tasks much faster.

concourse

(2) Concourse Task Inputs

Adding Task Inputs

In the previous section the only inputs to the task container was the image being used.

Since base images are static, relatively big, and slow things to create, Concourse supports inputs into tasks to pass in files/folders for processing.

Let's start by adding an 'inputs' block to the task and running it.

Example (MacOS):

```
cd demo-pipelines/02_task_inputs/
cat inputs_required.yml ##NOTICE THE ADDED INPUTS BLOCK
fly --target concourse-workshop execute --config inputs_required.yml
```

Tip: It should fail with '`error: missing required input `some-important-input`'` :)

Adding Task Inputs

While we told Concourse we wanted an input, we did not supply one!

Lets supply the actual input by tagging on `--input some-important-input=.` to the fly command

Example (MacOS):

```
fly --target concourse-workshop execute --config inputs_required.yml --input some-important-input=.
```

Adding Task Inputs

This time the fly execute command uploads the . (current) directory as an input to the container

Inside the execution container the input is made available at the path `some-import-input`

```
Output:
running ls -alR
.:
total 8
drwxr-xr-x    3 root    root        4096 Feb 27 07:27 .
drwxr-xr-x    3 root    root        4096 Feb 27 07:27 ..
drwxr-xr-x    1 501     20            64 Feb 27 07:27 some-important-input

./some-important-input:
total 12
drwxr-xr-x    1 501     20            64 Feb 27 07:27 .
drwxr-xr-x    3 root    root        4096 Feb 27 07:27 ..
-rw-r--r--    1 501     20           112 Feb 27 07:30 input_parent_dir.yml
-rw-r--r--    1 501     20           118 Feb 27 07:27 inputs_required.yml
-rw-r--r--    1 501     20            79 Feb 27 07:18 no_inputs.yml
```

Adding Task Inputs

The fly execute '`--input`' option can be removed if the current directory is the same name as the required input

The task '`input_parent_dir.yml`' contains an '`input 02_task_inputs`' which is also the current directory.

The following command will work and return the same results as before:

Example (MacOS):

```
fly --target concourse-workshop execute --config input_parent_dir.yml
```

concourse

(3) Concourse Task Scripts

Task Scripts

A common pattern is for Concourse tasks to run: complex shell scripts rather than directly invoking commands

Let's refactor and abstract what we did in the last part (running 'uname -a') into a script which we can run with the fly execute command

Start with the new bash script seen below:

Example (MacOS):

cd demo-pipelines/03_task_scripts

cat task_show_uname.sh

```
#!/bin/sh

uname -a
```

Task Scripts

Next let's create a new task yaml to execute using the current directory (where our script is located) as an input and the script we created as the run item.

Example (MacOS):

cd demo-pipelines/03_task_scripts

cat task_show_uname.yml

```
---
platform: linux
image_resource:
  type: docker-image
  source: {repository: busybox}
inputs:
- name: 03_task_scripts
run:
  path: ./03_task_scripts/task_show_uname.sh
```

Task Scripts

Notice that the bash script is executable (chmod +x task_show_uname.sh) and run it!

Example (MacOS):

```
cd demo-pipelines/03_task_scripts
fly --target concourse-workshop execute --config task_show_uname.yml ##RUNS THE CONCOURSE TASK
```

```
executing build 4 at https://ci.oskoss.com/builds/4
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 10240    0 10240    0     0  10240      0 --:--:-- --:--:-- --:--:--  123k
initializing
running ./03_task_scripts/task_show_uname.sh
Linux 795a1f41-e9c4-43f7-5f16-9f21dd1ed218 4.4.0-96-generic #119~14.04.1-Ubuntu SMP Wed Sep 13 08:40:48 UTC 2017 x86_64 GNU/Linux
succeeded
```

Tips:

Since input 03_task_scripts matches the current directory 03_task_scripts we did not need to specify fly execute --input 03_task_scripts=.

concourse

(4) Concourse Basic Pipeline

Basic Pipeline

Most all tasks that Concourse runs are within **'pipelines'**.

Pipelines allow you to model any continuous integration workflow!

- simple (unit → integration → deploy → ship)
- complex (testing on multiple infrastructures, fanning out and in, etc.)

3 Core Concepts

- **Tasks** (We already covered these)
- **Resources** - Entity that can be checked for new versions and updated to create new versions (Part 5)
- **Jobs** - Functions with inputs and outputs, that automatically run when new inputs are available.
  - Consists of **Tasks**

Basic Pipeline

Let's create a new pipeline that prints 'hello world'!

We start with a new Job and name it 'job-hello-world'
Inside the job we add a task with the image-resource/platform/run parameters.

Example (MacOS):

cd demo-pipelines/04_basic_pipeline

cat pipeline.yml

```yaml
Jobs:
  - name: job-hello-world
    public: true
    plan:
    - task: hello-world
      config:
        platform: linux
        image_resource:
          type: docker-image
          source: {repository: busybox}
        run:
          path: echo
          args:
          - hello world
```

Basic Pipeline

Next let's add the pipeline to concourse!

Example (MacOS):

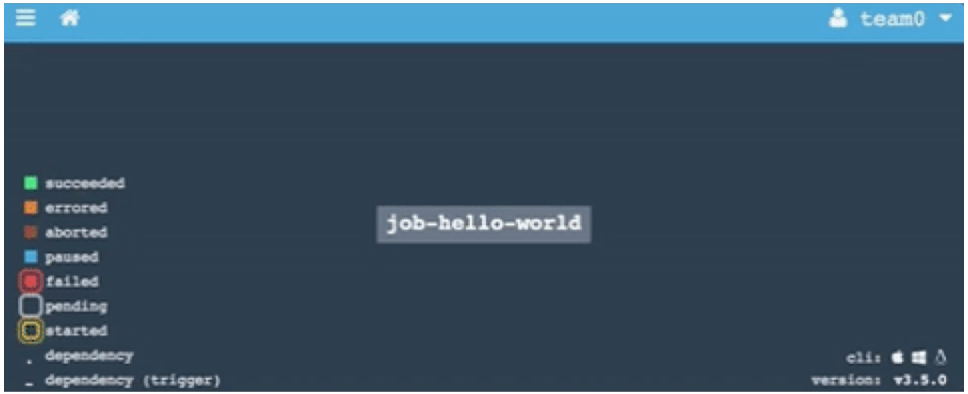
```
cd demo-pipelines/04_basic_pipeline
fly --target concourse-workshop  set-pipeline --config pipeline.yml --pipeline helloworld ##ADDS THE CONCOURSE PIPELINE
```

Tips: You can add --verbose to the above command to see the actual REST api commands being sent to concourse

Basic Pipeline

Now we just need to enable the pipeline and give it a run!

- Open a browser, navigate to https://ci.oskoss.com/ and login to the team your instructor provides along with the correct credentials.
- Click on the hamburger icon and press play. This enables the pipeline.
- Click the plus button and you should see a print out of 'hello world'!



concourse

(5) Concourse Resources

Using Resources

Concourse offers **no services** for storing/retrieving your data.

- No git repositories.
- No blobstores.
- No build numbers.

Every input and output must be provided **externally**. Concourse calls them "**Resources**".

- git
- s3
- Semver

To show how resources can be used, let's take the task we had embedded into a 'pipeline.yml' and extract it to a git repo resource!

Using Resources

Inside the 'pipeline.yml' we now have a resources section which contains a git repository where the task script lives.

We also now add a 'get:' step to the hello world job which git clones the repository we specified earlier.

Finally we specify the task yaml file to run.

Example (MacOS):

cd demo-pipelines/05_pipeline_task_hello_world

cat pipeline.yml

```
resources:
- name: resource-tutorial
  type: git
  source:
    uri: https://github.com/starkandwayne/concourse-tutorial.git
Jobs:
  - name: job-hello-world
    public: true
    Plan:
    - get: resource-tutorial
    - task: hello-world
      file: resource-tutorial/01_task_hello_world/task_hello_world.yml
```

Using Resources

Now we will update the previous helloworld pipeline with a new one!

Example (MacOS):

```
cd demo-pipelines/05_pipeline_task_hello_world
fly --target concourse-workshop  set-pipeline --config pipeline.yml --pipeline helloworld ##UPDATE CONCOURSE PIPELINE
```

Tips: The output will show the **delta** between the two pipelines and request confirmation. Type y!

Using Resources

Finally, check out the updated pipeline in the UI at https://ci.oskoss.com

Notice the in-progress or newly-completed job-hello-world job UI has three sections:

- Preparation for running the job - collecting inputs and dependencies
- resource-tutorial resource is fetched (from github)
- hello-world task is executed

Using Resources

There are benefits and downsides to abstracting tasks into YAML files outside of the pipeline.

+ The task keeps in sync with the primary input resource (for example, a software project with tasks for running tests, building binaries, etc).
+ The 'pipeline.yml' file can get long and it can be hard to read and comprehend all the YAML. When extracting tasks into separate files shrinks this.
- Comprehension of pipeline behavior is potentially reduced since the 'pipeline.yml' no longer explains exactly what commands will be invoked.
- Extracting inline tasks into files, fly set-pipeline is no longer the only step to updating a pipeline.

concourse

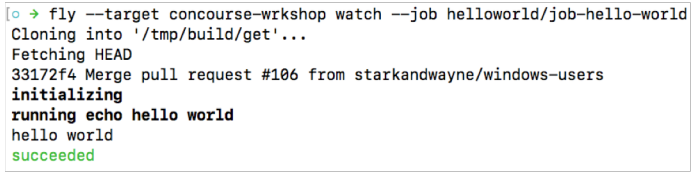
(6) Concourse CLI (Jobs).pptx

CLI (Jobs)

While viewing output through the UI is a great quick glance, we can check the output via CLI as well.

Example (MacOS):

fly --target concourse-workshop watch --job helloworld/job-hello-world ##CHECK JOB OUTPUT



Tips: --build NUM option allows you to see the output of a specific build number, rather than the latest build output.

CLI (Jobs)

You can see the results of recent builds across all pipelines with fly builds:

Example (MacOS):

fly --target concourse-workshop builds ##LIST ALL BUILDS

```
[○ → fly --target concourse-wrkshop  builds
id  pipeline/job               build  status     start                      end                        duration
8   helloworld/job-hello-world  2      succeeded  2017-10-22@21:10:13-0500  2017-10-22@21:10:16-0500  3s
7   helloworld/job-hello-world  1      succeeded  2017-10-22@11:37:12-0500  2017-10-22@11:37:13-0500  1s
```

concourse

(7) Concourse Triggers

Triggers

There are three ways for a job to be triggered:

- Clicking the + button on the web UI of a job (as we did in previous sections)
- Input resource triggering a job (see part 8)
- `fly --target target trigger-job --job pipeline/jobname` command

Example (MacOS):

```
fly --target concourse-workshop trigger-job --job helloworld/job-hello-world ##MANUALLY TRIGGER JOB
```

Triggers

Whilst the job is running, and after it has completed, you can then watch the output in your terminal using `fly watch`

```
○ → fly --target concourse-wrkshop watch --job helloworld/job-hello-world
initializing
running echo hello world
hello world
succeeded
```

concourse

(8) Concourse Resource Triggers

Resource Triggers

The primary way that Concourse jobs will be triggered to run will be by resources changing.

- `git` repo has a new commit? Run a job to test it.
- GitHub project cuts a new release? Run a job to pull down its attached files and do something with them.

By default, including `get: my-resource` in a build plan does not trigger its job.

To mark a fetched resource as a trigger add `trigger: true`

Resource Triggers

If you want a job to trigger every few minutes then there is the time resource.

Lets add the time resource to the 'pipeline.yml' and utilize it to trigger the pipeline.

Example (MacOS):

cd demo-pipelines/08_triggers

cat pipeline.yml

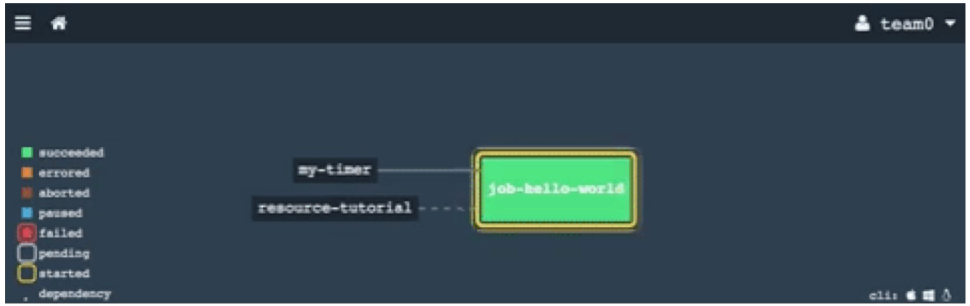
```
resources:
- name: resource-tutorial
  type: git
  source:
    uri: https://github.com/starkandwayne/concourse-tutorial.git
- name: my-timer
  type: time
  source:
    interval: 2m
```

```
Jobs:
  - name: job-hello-world
    public: true
    Plan:
    - get: resource-tutorial
    - get: my-timer
      trigger: true
    - task: hello-world
      file: resource-tutorial/01_task_hello_world/task_hello_world.yml
```

Resource Triggers

Update the pipeline and visit the pipeline dashboard https://ci.oskoss.com and wait a few minutes and eventually the job will start running automatically.

```
cd demo-pipelines/08_triggers
fly --target concourse-workshop  set-pipeline --config pipeline.yml --pipeline helloworld ##UPDATE CONCOURSE PIPELINE
```



Resource Triggers

The dashboard UI makes non-triggering resources distinct with a hyphenated line connecting them into the job.

Triggering resources have a full line.

concourse

(9) Concourse Destroying Pipelines

Destroying Pipelines

At this point our last pipeline will continue to trigger every 2 minutes….forever!

Lets destroy the pipeline. This will ERASE all its build history!

Example (MacOS):

```
fly --target concourse-workshop destroy-pipeline --pipeline helloworld
```

concourse

(10) Concourse Unit Test Pipeline

Unit Test Pipeline

We have covered a lot up to this point

Let's bring everything together into a real pipeline that will run unit tests on a sample golang web application server

simple-go-web-app has some basic unit tests, and in order to run those tests inside a pipeline we need:

- a task image that contains any dependencies
- an input resource containing the task script that knows how to run the tests
- an input resource containing the application source code

Unit Test Pipeline

Since we are compiling a golang app, lets use use the golang:1.6-alpine image from:

https://hub.docker.com/_/golang/

(see https://imagelayers.io/?images=golang:1.6-alpine for size of layers)

Example (MacOS):

cd demo-pipelines/10_job_inputs

cat task_run_tests.yml

```
image_resource:
  type: docker-image
  source: {repository: golang, tag: 1.6-alpine}
```

Unit Test Pipeline

We need our resources:

- Golang code we are testing located -> https://github.com/cloudfoundry-community/simple-go-web-app.git
- Test script to execute located -> https://github.com/starkandwayne/concourse-tutorial.git

Example (MacOS):

cd demo-pipelines/10_job_inputs

cat pipeline.yml

```
resources:
- name: resource-tutorial
  type: git
  source:
    uri: https://github.com/starkandwayne/concourse-tutorial.git
- name: resource-app
  type: git
  source:
    uri: https://github.com/cloudfoundry-community/simple-go-web-app.git
```

Unit Test Pipeline

Let's give it a go!

Example (MacOS):

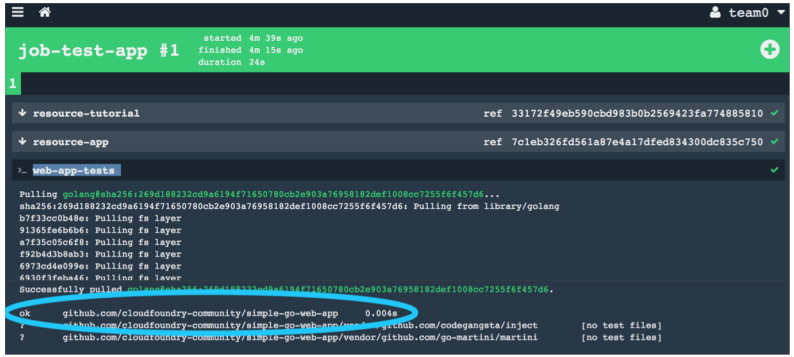
```
cd demo-pipelines/10_job_inputs
fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline simple-app --non-interactive
fly unpause-pipeline --target concourse-workshop --pipeline simple-app
```

Unit Test Pipeline

The pipeline (https://ci.oskoss.com) automatically starts this time since we un-paused the pipeline via the fly CLI.

Notice the job pauses on the first run at web-app-tests task because it is downloading the golang:1.6-alpine image for the first time.

If all was successful you should see the following:



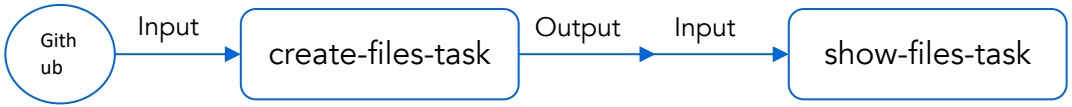
concourse

(11) Concourse Task Outputs

Task Outputs

Some tasks you'll create in Concourse will want to pass output to another task for further processing.

In our pipelines thus far we have gotten all of our resources from github, let's add another task that takes its resource from the output of the previous task.



Task Outputs

In the first task we now add both inputs: and outputs: tags

Example (MacOS):

cd demo-pipelines/11_task_outputs_to_inputs

cat create_some_files.yml

```
platform: linux
image_resource:
  type: docker-image
  source: {repository: busybox}
inputs:
- name: resource-tutorial
outputs:
- name: some-files
run:
  path: resource-tutorial/11_task_outputs_to_inputs/create_some_files.sh
```

Task Outputs

While Concourse creates a folder for us under the name we give in the `outputs:` tag, it is the responsibility of the `run:` script to place artifacts there

Example (MacOS):

```
cd demo-pipelines/11_task_outputs_to_inputs
cat create_some_files.sh
```

```
#!/bin/sh

mkdir some-files
echo "file1" > some-files/file1
echo "file2" > some-files/file2
echo "file3" > some-files/file3
echo "file4" > some-files/file4

ls some-files/*
```

Tips: The mkdir should **fail** when we run the pipeline since concourse already has made the directory!

Task Outputs

Subsequent tasks now will have the some-files folder available, notice the show_files.sh script lists the files from the previous task

Example (MacOS):

cd demo-pipelines/11_task_outputs_to_inputs

cat show_files.sh

```
#!/bin/sh

ls some-files/*
```

Finally let's update and unpause our pipeline

Example (MacOS):

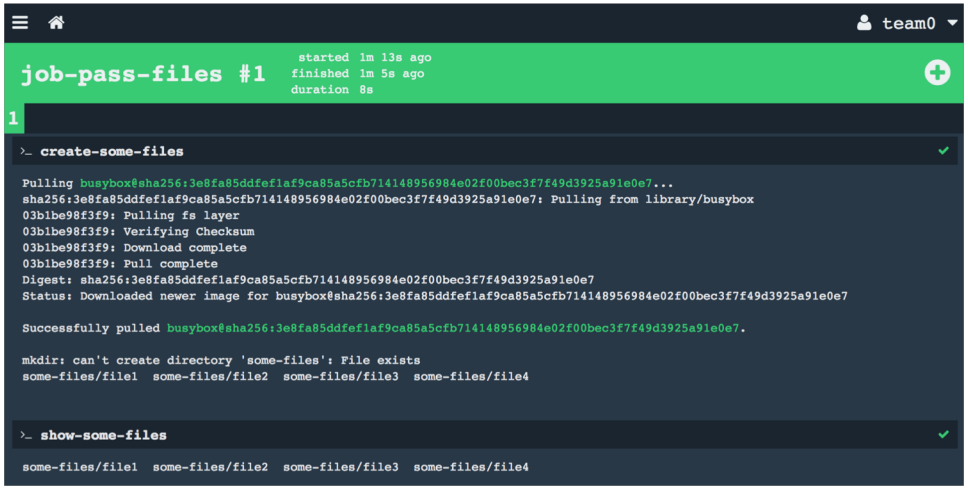
cd demo-pipelines/11_task_outputs_to_inputs

fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline pass-files --non-interactive

fly unpause-pipeline --target concourse-workshop --pipeline pass-files

Task Outputs

If we manually trigger the pipeline (https://ci.oskoss.com) you should see files being created by one task and the following task listing them!



Tips: The pipeline only shows that two tasks are to be run in a specific order. It does not indicate that show-files/ is an output of one task and used as an input into the next task.

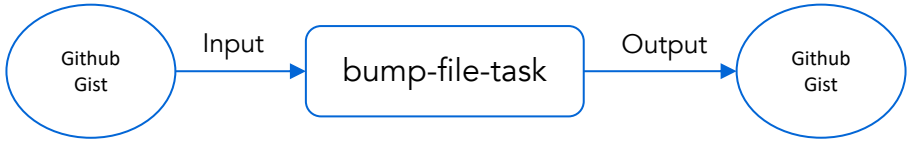
concourse

(12) Concourse Publishing Outputs

Publishing Outputs

While tasks can pass output to other tasks, they can also push output to the external world.

Lets create a new pipeline that pushes updates to a github gist.



Publishing Outputs

Let's start by updating the pipeline and unpausing it

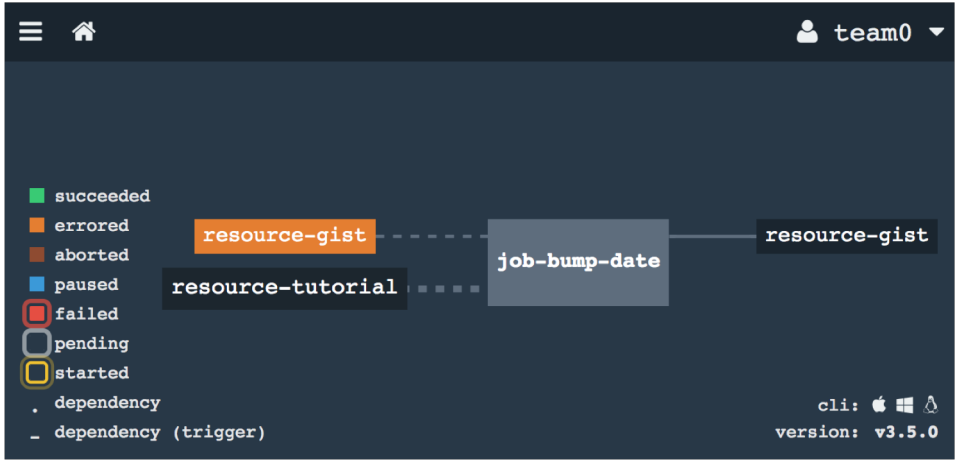
Example (MacOS):

```
cd demo-pipelines/12_publishing_outputs

fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline publishing-outputs --non-interactive

fly unpause-pipeline --target concourse-workshop --pipeline publishing-outputs
```

Publishing Outputs

You may notice now that the pipeline has an orange box.

This indicates that the input resource is erroring!



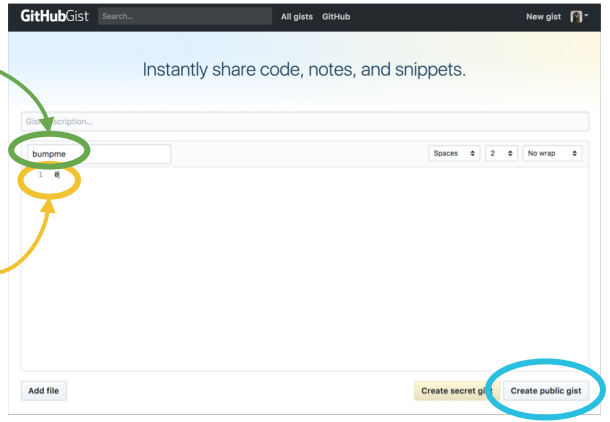
Publishing Outputs

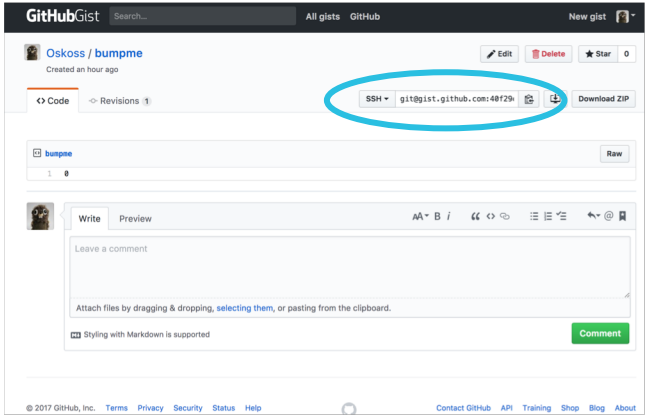
We forgot to specify the gist we want to update…

Lets solve it by creating a gist and grabbing the ssh link for it

Name of Gist:
bumpme

Version number:
0





Publishing Outputs

Now paste it into our pipeline.yml

SSH URL of Gist:

```
[○ → cat ~/workspace/Workshops-2.0/IT\ Operator/Automation\ Track/Pipelines/12_pu]
blishing_outputs/pipeline.yml
---
resources:
- name: resource-tutorial
  type: git
  source:
    uri: https://github.com/starkandwayne/concourse-tutorial.git
- name: resource-gist
  type: git
  source:
    branch: master
    uri: git@gist.github.com:40f29d2005e30c5df16ab100d05007b6.git
    private_key: |
      -----BEGIN RSA PRIVATE KEY-----
      MIIJKAIBAAKCAgEAt+6xorroA/YM33PCzwJFO5pweOwsemcaqTLIyy1vSlfIw6ix
      QFp4HTI74OCGwjfoDBKUOGSRXt1dAAFSpw4zpyTdjzw79OnRAhBbINV73cYFGD5R
      7rVTiERXzCDs6xIyXL6yz/1vIMKTtRyppYBeJDyg36UIy/QGfn1txTIHCsKmztLV
      7q1N3l5o1345wUmAg8G0hvisu6KoJpcdD5YJmsr1R5g31dZOZlx7ttplRqS31RY2
      lKb8aG7Pfrfbw9700RhUCvxF6q+LG1C7vud//acXxbxh0AZD6aHp5PmlkL4XlCZq
      2hSm8BVv/CoF3LoUHjF04vRydHz/C7AR966A7NWp9k0DYR+WBeQ6KLAWiTfEObmc
      2NnC4llvgZxlRy0KH9EOXUACdXgedhFHzNQoyeei9tB7+sQJBw3bWv3oN5dASBii
      H/H2QJDv1KU5Fl/KNv9lG5F1MWeNT/nQOf9iIJi4ox59sId94c6iqae42xeiyyGu
      puDO9qDH01Jt4coiqSesPRaX/NTZPU2086wkmutIus8TTtr21Juw2NEwYSuvqFFqZ
      L9ngE+m58+S61QXcHuCMJnp92xA0IUuFnlMjrxvo0BLIfYCzjU94LRV8tTtTpw2c
      8qQpXv1rYtK5UyZ0TQW0FdsigcCzQZoVeUTiLXwqZleqCRiXcBNnVsjFdm0CAwEA
      AQKCAgATKncuGsXEa+YpfpmFEpp+PHiN0SZNVuMLUHRl34UiPQV5heF6cDLl7+8X
      Xq4MHOE8KM9zFN+k//IJ5xliuPMe7pN7sm4Y9ihGS8gDItu2X0T2or/M3O4oW96+
      MYwebpX63hAp5iGnSz8y9xqiWT72Da9BvqOtBO+MTKucibr8Hw5crNGH/l0H19Sa
      59dC0dLnZ87U5aKyJp/Y7ZQkpKC5lX57iFwBF0SyG970TcuR0eYp+ReU49WXWDyc
      2rG2tkvSYfOX9XOfoc0f+oilut+2Yavzec7Da3O2K54AsxbSf0xbrqkhslc22zUB
```

Publishing Outputs

Finally paste your github ssh key into your pipeline.yml and update the pipeline!

SSH Key:

Example (MacOS):

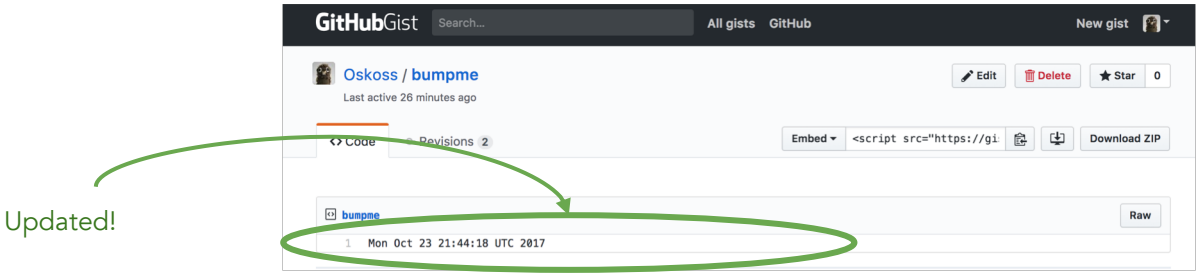
cd demo-pipelines/12_publishing_outputs

fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline publishing-outputs --non-interactive

Publishing Outputs

Revisit the dashboard UI and the orange resource will change to black if it can successfully fetch the new git@gist.github.com:XXXX.git repo

After running the job-bump-date job, refresh your gist:



Example (MacOS):

cd demo-pipelines/12_publishing_outputs

fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline publishing-outputs --non-interactive

Publishing Outputs

If we take a look at the bump-timestamp-file.yml notice it has a new output updated-gist

```
outputs:
 - name: updated-gist
```

The task runs the bump-timestamp-file.sh script

```
git clone resource-gist updated-gist
cd updated-gist
echo $(date) > bumpme
git config --global user.email "nobody@concourse.ci"
git config --global user.name "Concourse"
git add .
git commit -m "Bumped date"
```

Publishing Outputs

Finally, there is a new `put:` step and resource to send out artifacts to the git gist in our `pipeline.yml`

```
- put: resource-gist
  params: {repository: updated-gist}
```

```
- name: resource-gist
  type: git
  source:
    branch: master
    uri: git@gist.github.com:40f29d2005e30c5df16ab100d05007b6.git
    private_key: |
      -----BEGIN RSA PRIVATE KEY-----
      MIIJKAIBAAKCAgEAt+6xorroA/YM33PCzwJFO5pweOwsemcaqTLIyy1v
```

concourse

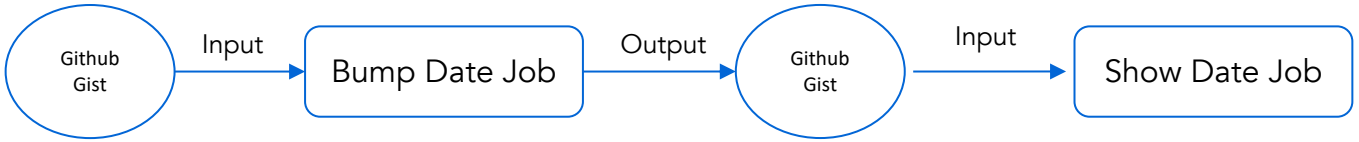
(13) Concourse Complete Pipeline

Complete Pipeline

The time has come! Up to this point our pipelines have only had a single job. For all their wonderfulness, they haven't yet felt like actual pipelines.

Jobs passing results between jobs. This is where Concourse shines even brighter.

Lets create a new pipeline that updates the date and then shows the date in another job.



Complete Pipeline

Let's start by updating the previous pipeline.yml by adding a second job:

```
- name: job-show-date
  plan:
  - get: resource-tutorial
  - get: resource-gist
    passed: [job-bump-date]
    trigger: true
  - task: show-date
    config:
      platform: linux
      image_resource:
        type: docker-image
        source: {repository: busybox}
      inputs:
        - name: resource-gist
      run:
        path: cat
        args: [resource-gist/bumpme]
```

Example (MacOS):

cd demo-pipelines/13_pipeline_jobs

cat pipeline.yml

Complete Pipeline

Now update the Concourse Pipeline:

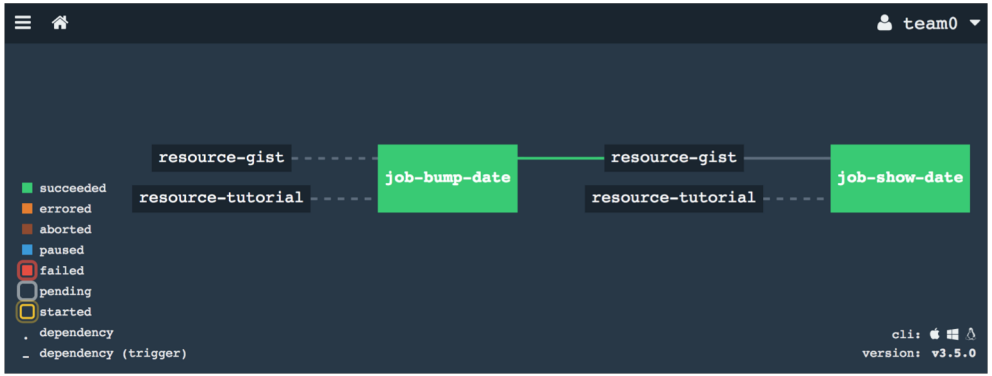
Example (MacOS):

```
cd demo-pipelines/13_pipeline_jobs
fly --target concourse-workshop set-pipeline --config pipeline.yml --pipeline publishing-outputs --non-interactive
```

Complete Pipeline

The latest resource-gist commit fetched down in job-show-date will be the exact commit used in the last successful job-bump-date job.

If you manually created a new git commit in your gist and manually ran the job-show-date job it would continue to use the previous commit it used, and ignore your new commit.



This is the power of pipelines.

concourse

(14) Concourse Hiding Secrets

Hiding Secrets

Pipelines often need access to secured resources, yet storing credentials inside the `pipeline.yml` file (as we did in previous parts) makes it difficult to share.

Instead Concourse provides the parameters abstraction layer, because not everyone needs nor should have access to the shared secrets.

Parameters inside the `pipeline.yml` file are written in this notation: `{{parameter}}`.

Hiding Secrets

Parameters are all mandatory!

For an example try (MacOS):

```
cd demo-pipelines/14_parameters
fly set-pipeline --target tutorial --config pipeline.yml --pipeline publishing-outputs --non-interactive
```

You should see an error output similar to the following since the new `pipeline.yml` has parameters and we did not specify them

```
failed to evaluate variables into template: 2 error(s) occurred:

* unbound variable in template: 'gist-url'
* unbound variable in template: 'github-private-key'
```

Hiding Secrets

To specify secrets create a credentials.yml file with keys gist-url and github-private-key.

The values come from your previous pipeline.yml files:

```
gist-url: git@gist.github.com:xxxxxxx.git
github-private-key: |
  -----BEGIN RSA PRIVATE KEY-----
  MIIEpQIBAAKCAQEAuvUl9YUlDHWBMVcuu0FH9u2gSi83PkL4o9TS+F185qDTlfUY
  fGLxDo/bn8ws8B88oNbRKBZR6yig9anIB4Hym2mSwuMOUAg5qsA9zm5ArXQBGoAr
  ...
  iSHcGbKdWqpObR7oau2LIR6UtLvevUXNu80XNy+jaXltqo7MSSBYJjbnLTmdUFwp
  HBstYQubAQy4oAEHu8osRhH1VX8AR/atewdHHTm48DN74M/FX3/HeJo=
  -----END RSA PRIVATE KEY-----
```

Hiding Secrets

To pass in your credentials.yml file use the --load-vars-from or -l options:

Example (MacOS):

```
cd demo-pipelines/14_parameters
fly set-pipeline --target concourse-workshop --config pipeline.yml --pipeline publishing-outputs --non-interactive --load-vars-from <SUPER_SECRET_DIR>/credentials.yml
```