Building pipelines in Concourse using jsonnet

Mario Fernandez

ThoughtWorks

A problem to solve

Provisioning a ton of infrastructure

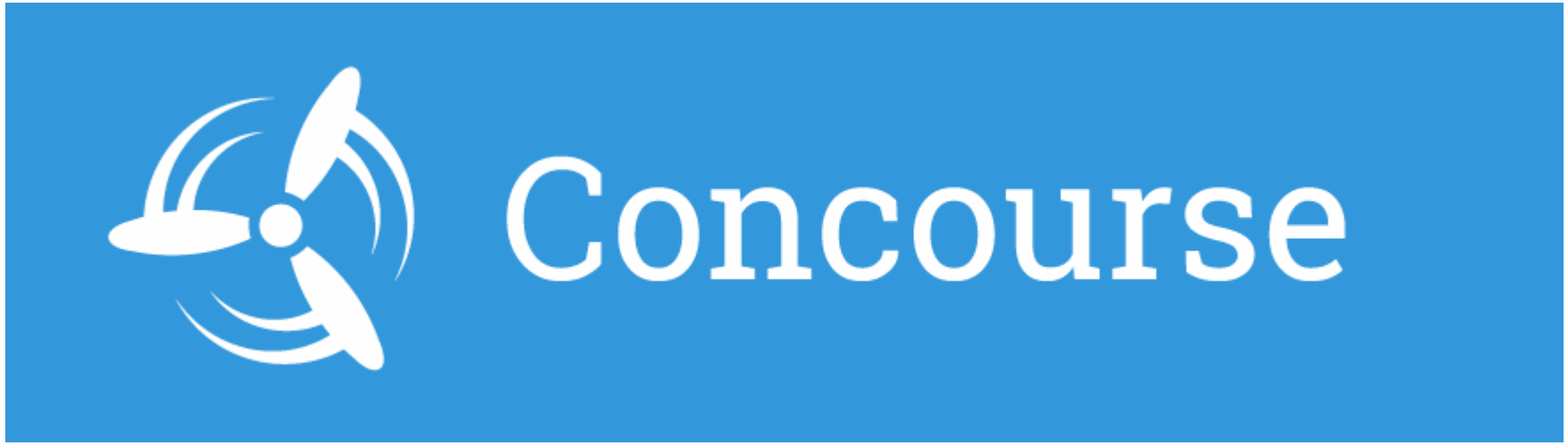
```
modules
├── access-from-other-accounts
├── access-to-other-accounts
├── application-environment-1
├── application-environment-2
├── cluster-{backup,management}
├── connected-hosted-zone
├── dispatcher-vpc
├── domain
├── functional-area
├── functional-area-{access,runtime}
├── logging
├── monitoring
├── operations
├── private-egress
├── private-egress-acme
├── private-ingress-{app,dispatcher}
└── system-services
```

Comprehensive

multiple environments

multiple regions

Multiple products



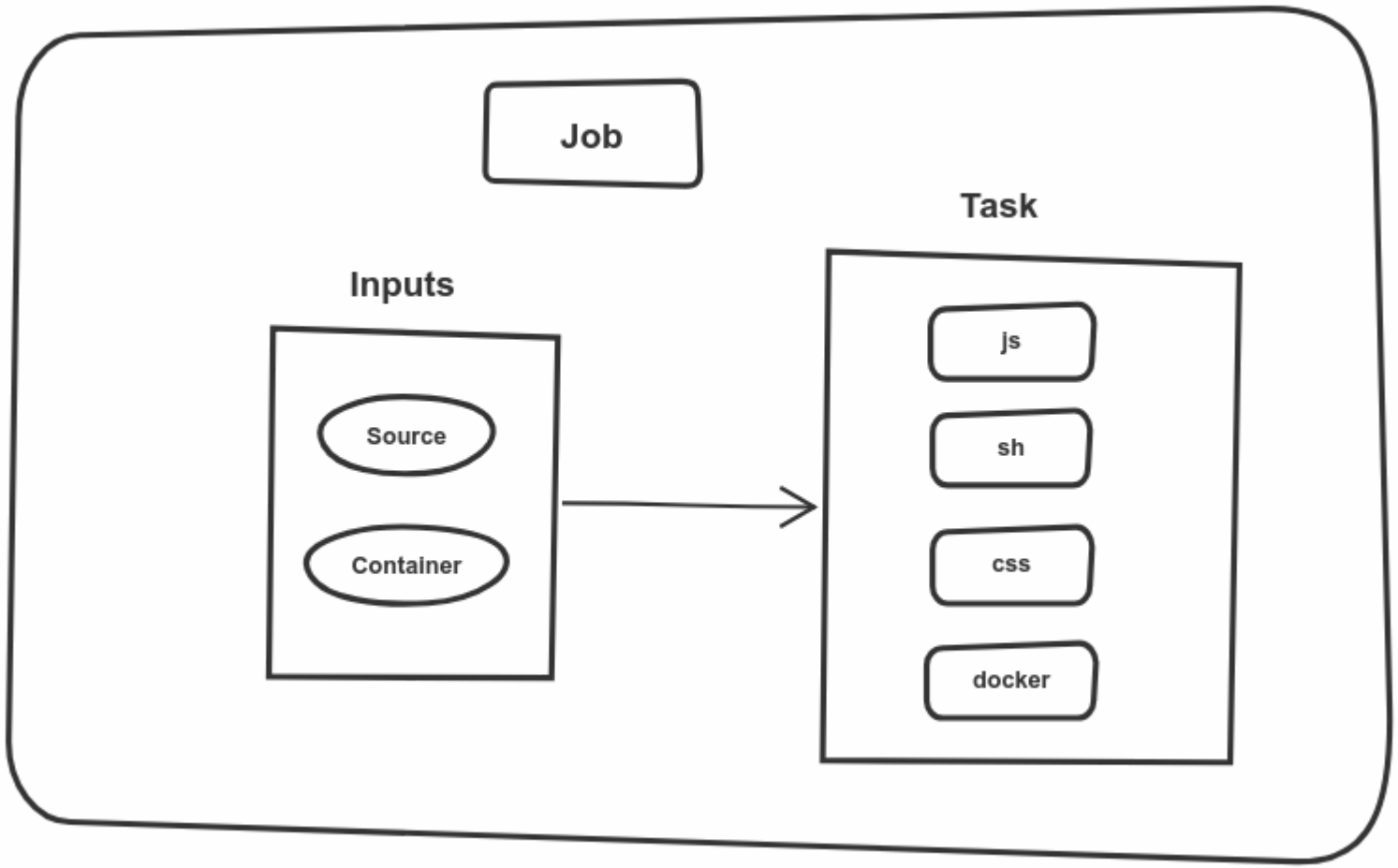
thoughtworks.com/radar/techniques/pipelines-for-infrastructure-as-code

The issue

YAML is really verbose

Like, really

Concourse doesn't help, either


Job
Task
Inputs
Source
Container
js
sh
css
docker

```yaml
- name: lint
  serial: true
  plan:
  - in_parallel:
    - get: git
      passed: [prepare]
      trigger: true
    - get: dev-container
      passed: [prepare]
  - in_parallel:
    - task: lint-sh
      image: dev-container
      params:
        <<: *common-params
        TARGET: sh
      file: git/pipeline/tasks/linter/task.yml
```

Our ability to manage duplication is limited

```
  - in_parallel:
    - task: lint-sh
      image: dev-container
      params:
        <<: *common-params
        TARGET: sh
      file: git/pipeline/tasks/linter/task.yml
```

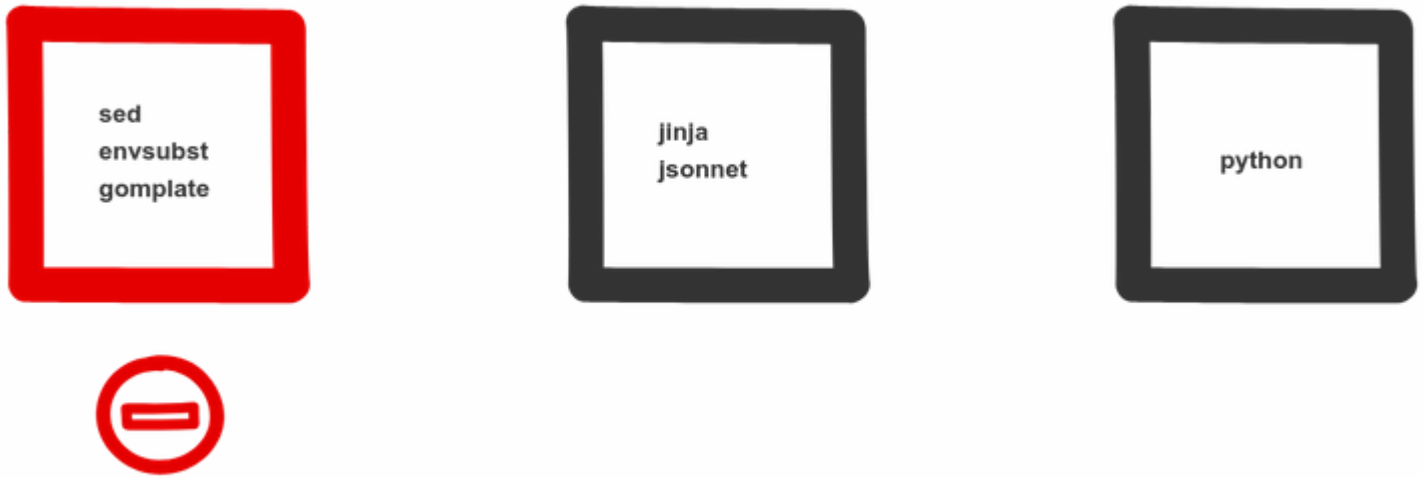
```
  - in_parallel:
    - task: lint-sh
      image: dev-container
      params:
        <<: *common-params
        TARGET: sh
      file: git/pipeline/tasks/linter/task.yml
```

YAML overdose!

What are our options?

Dare I say, templating?


sed
envsubst
gomplate
jinja
jsonnet
python


sed
envsubst
gomplate
jinja
jsonnet
python

Templating in YAML

APR 2019

HOLD

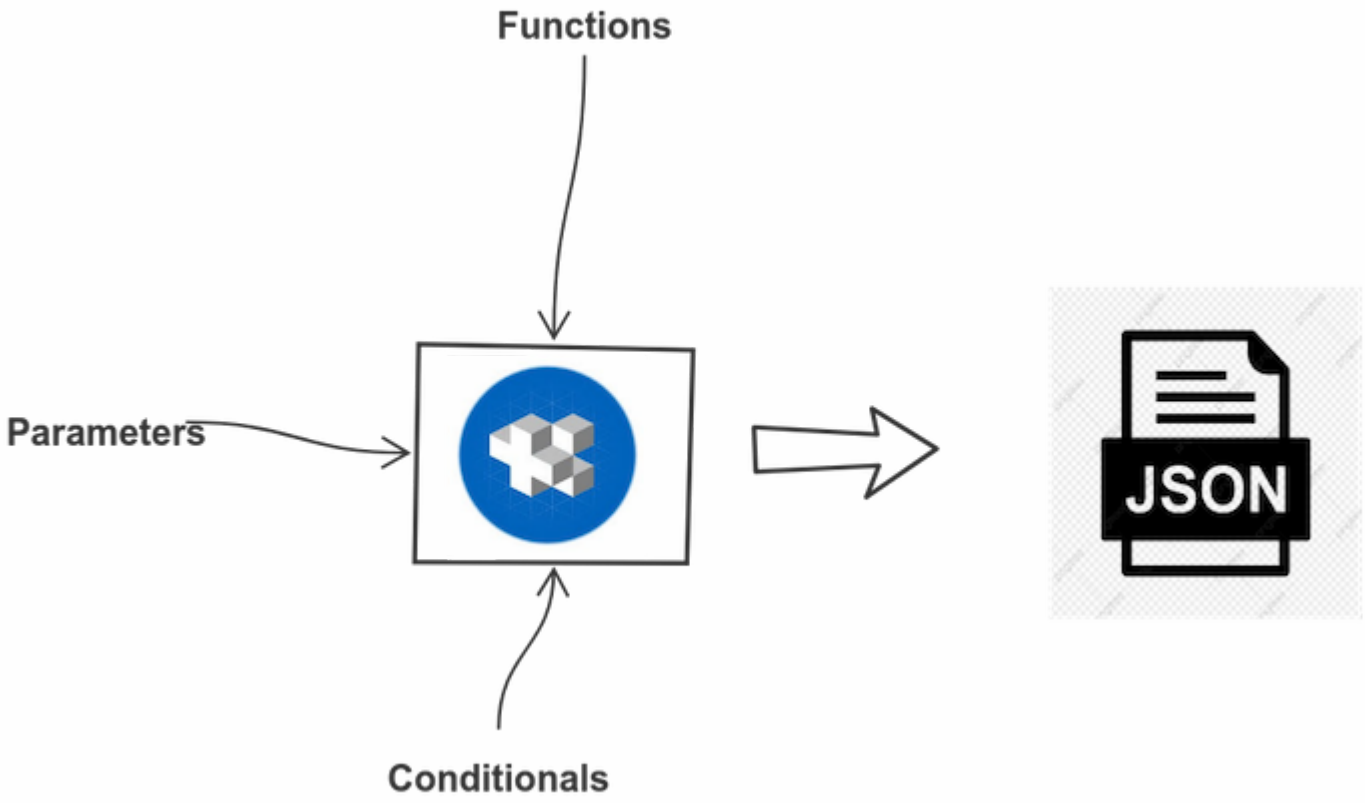
As infrastructures grow in complexity, so do the configuration files that define them. Tools such as **AWS CloudFormation**, **Kubernetes** and **Helm** expect configuration files in JSON or YAML syntax, presumably in an attempt to make them easy to write and process. However, in most cases, teams quickly reach the point where they have some parts that are similar but not quite the same, for example, when the same service must be deployed in different regions with a slightly different setup. For such cases tools offer **templating in YAML** (or JSON), which has caused a huge amount of **frustration with practitioners**. The problem is that the syntax of JSON and YAML requires all sorts of awkward compromises to graft templating features such as conditionals and loops into the files. We recommend using an API from a programming language instead or, when this is not an option, a templating system in a programming language, either a general-purpose language such as Python or something specialized such as **Jsonnet**.

> We recommend using an API from a programming language, or, a templating system

Jsonnet

jsonnet.org/

A data templating language for app and tool developers


Functions
Parameters
Conditionals
JSON

YAML happens to be a superset of JSON

Abstracting building blocks

```
local Parallel(tasks) = {
  in_parallel: tasks
}
```

```
local Job(name, serial = true, plan = []) = std.prune({
  name: name,
  serial: serial,
  plan: plan
})
```

```
DockerResource(name,
               repository,
               tag = 'latest', allow_insecure = false) = {
  name: name,
  type: 'docker-image',
  source: {
    repository: repository,
    tag: tag
  } + (
    if allow_insecure then {
      insecure_registries: [std.split(repository, '/')[0]]} else {}
  ),
}
```

github.com/sirech/concourse-jsonnet-utils

Building your own DSL

Reflect your conventions and idioms in code

```
local source = 'git';
local container = 'dev-container';

local Inputs(dependencies = []) = concourse.Parallel(
  [concourse.Get(s, dependencies = dependencies)
    for s in [source, container]]
);
```

```
local Task(name, file = name, image = container, params = {}) = {
  task: name,
  image: image,
  params: { CI: true } + params,
  file: '%s/pipeline/tasks/%s/task.yml' % [source, file]
};
```

Back to the original example

```yaml
- name: lint
  serial: true
  plan:
  - in_parallel:
    - get: git
      passed: [prepare]
      trigger: true
    - get: dev-container
      passed: [prepare]
  - in_parallel:
    - task: lint-sh
      image: dev-container
      params:
        <<: *common-params
        TARGET: sh
      file: git/pipeline/tasks/linter/task.yml
```

```
concourse.Job('lint', plan = [
  Inputs('prepare'),
  concourse.Parallel(
    [Task('lint-%s' % lang, 'linter', params = { TARGET: lang })
      for lang in ['sh', 'js', 'css', 'docker']]
  )
]),
```

github.com/sirech/example-concourse-pipeline

Scaling up to multiple pipelines

First goal

Generate one pipeline programmatically

Next goal

Generate many pipelines

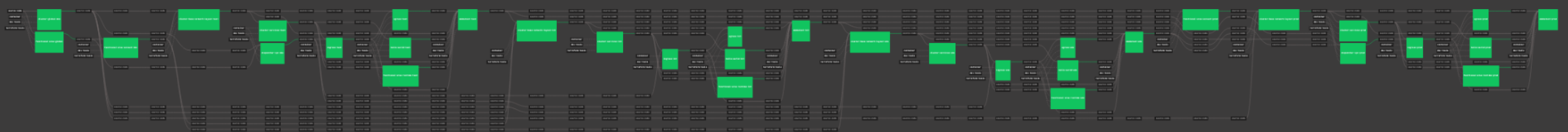
```
local config = std.extVar('CONFIG')
```

```
# Convert the config to JSON and merge it, so that values are overriden
CONFIG="$(yq -r '. * .' ../product_defaults.yaml product.yaml \
          | jq -s 'add')"
```

```yaml
product_name: new-product
version: master

clusters:
  test:
    eu-west-1:
  prod:
    eu-west-1:
    us-east-1:

accounts:
  test: "product"
  prod: "product-prod"
```

Some results

```
local EnvironmentJobs(env) =
  local regions = std.objectFields(config.clusters[env]);
  local params = { ENV: env };
  local services = ['system-services', 'monitoring', 'logging'];

  local modules = [
    Job('cluster-services-%s' % [env],
        'cluster-global-dns',
        tasks=[TaskPerRegion(module, regions, params)
               for module in services]),
  ];

  modules + [
    Job('smoketest-%s' % [env],
        ['%s' % [module] for module in [job.name for job in modules]],
        tasks=[TaskPerRegion('smoketest', regions, params)]),
  ]
```



Generated	LOC
pipeline.yaml	3102

Jsonnet	LOC
pipeline.jsonnet	94
concourse.libsonnet	54
builders.libsonnet	40
environments.libsonnet	19

hceris.com/templating-concourse-pipelines-with-jsonnet/