Building EVOLUTIONARY infrastructure
Some visuals found from the *Vault of the Atomic Space Age*
http://bit.ly/2KBTbAd
Why?

Ongoing pressure to deliver while requirements change
Inconsistent path to production

DEV

TEST

PROD
Snowflake production
Continuous, reliable delivery of changes to all your stuff

DEV

TEST

PROD
"Since we can't avoid change, we need to exploit it"

Building Evolutionary Architectures
Neal Ford, Rebecca Parsons, Pat Kua
Speed or Quality?
Careful — Fast

Slow — Careless
Move fast and fix things
Careful  Fast

Slow

Careless
Prioritize speed over correctness
Slow

Careful

Prioritize correctness over speed of change

Careless

Fragile mess

ThoughtWorks
Careful

Prioritize correctness over speed of change

Fragile mess

Careless

Time pressures
Over-engineering
Technical debt
Barriers to improvement
Optimize for change

Build the simplest thing

Build quality in

Fast, frequent feedback

Fast

Agile, Lean, DevOps

Careful

Careless

Slow
Infrastructure Tools and Technology
Dynamic infrastructure platform

- Development Tooling
- Change delivery services
- Infrastructure management tools
- Application runtime platform
- Application Packaging and Deployment

Operational management services

ThoughtWorks®
Dynamic infrastructure platform

A pool of resources
Provision resources

• On demand

• Programmatically
Variations of compute resources

- Physical Servers
- Virtual Machines
- Containers
- Serverless (FaaS)
Development Tooling

Application Packaging and Deployment

Application runtime platform

Change delivery services

Infrastructure management tools

Dynamic infrastructure platform

Operational management services
Server configuration tool

Configuration tool

Configuration definitions

Server instance

(Ansible, Chef, Puppet, Saltstack, ...)

ThoughtWorks®
Server image creation

Base server image

Image definition

Server configuration definitions

Custom image

(Packer, ...)

ThoughtWorks®
Concept: Infrastructure Stack
Infrastructure stack

A collection of infrastructure resources provisioned and updated as a unit
A stack may include the infrastructure for a single application
Or, multiple applications might run in a single stack.
Or the parts of one application could be spread across several stacks
A stack definition is code used by a tool to provision stack instances.
A single stack definition may be used to manage one or more stack instances.
Multiple environments
Starting small – single stack instance

One or two people working on it,

... the system is fairly simple,

... and there are few users, with low stakes
And then ...

More people join the team,

... the system becomes more complex,

... and more people rely on the system to be working.
Multiple environments

TEST

QA

PRODUCTION
Multiple environments defined in one stack instance
But the blast radius for a change is too wide.
Separate stack definition project for each stack instance

TEST

QA

PRODUCTION
Single definition project template, used to create a separate stack instance for each environment.
Pipelines for infrastructure
Automatically test every change before applying it
Promote changes to environments using a pipeline.
Stack tool is only run by pipeline agents
Why not apply changes from our laptops?

Our working environments may not be consistent
Multiple people may clash
We may shortcut our process
We may neglect to run all appropriate tests
"Local" pre-commit sandbox sandboxes

Nita's Instance

Freddy's Instance

Alex's Instance
Parameterizing stacks
Parameters per stack instance

```yaml
server "web" {
  name = "web-\${env}\"
  type = "\${type}\"
}
```

- **web-test**
  - env = "test"
  - type = "t2.tiny"

- **web-qa**
  - env = "qa"
  - type = "t2.large"

- **web-prod**
  - env = "prod"
  - type = "t2.large"
Parameters in source project folders

- infra
  - web_server.tf
  - subnet.tf
  - vpc.tf
- environments
  - test.tfvars
  - qa.tfvars
  - prod.tfvars
Parameters in pipeline job definitions

BUILDS

APPLY TO TEST

APPLY TO QA

APPLY TO PROD
Parameters from configuration registry

- **web-test**: envs/test/type → t2.tiny
- **web-qa**: envs/qa/type → t2.large
- **web-prod**: envs/prod/type → t2.large
Recommendation

Minimize the configurable surface area to reduce variation between environments
Infrastructure testing
An effective testing regime drives a clean, decoupled, evolvable system design.
Fitness functions

Define the important characteristics of your system, how to measure these, and how to assess progress
Example: *our-app* stack

- **Infrastructure stack**
- **Application server**
- **JDK**
- **Tomcat**
- **OurApp.war**
Example pipeline for our-app

```
BUILD OUR-APP -> TEST OUR-APP -> TEST ALL -> QA -> PROD
```

- **TEST JDK COOKBOOK**
- **TEST TOMCAT COOKBOOK**
- **TEST APP SERVER ROLE**
- **BUILD APP SERVER AMI**
- **VALIDATE INFRA STACK**
- **TEST INFRA STACK**
APPLICATION STAGES:

Compile, unit tests, container tests, packaging, etc. (run on build agents)

JUnit, Rspec, etc.
SERVER PACKAGES:
Validate, test individual package configurations in isolation

TEST JDK COOKBOOK
TEST TOMCAT COOKBOOK

TEST APP SERVER ROLE
BUILD APP SERVER AMI
TEST INFRA STACK

SERVER ROLE:
Test aggregated packages

TEST ALL
QA
PROD

Typically run on build agents, perhaps containerized

test-kitchen, serverspec, inspec, puppet-rspec, testinfra, etc.
BUILD OUR-APP → TEST OUR-APP → TEST ALL → QA → PROD

TEST JDK COOKBOOK

TEST TOMCAT COOKBOOK

TEST APP SERVER ROLE

BUILD APP SERVER AMI

VALIDATE INFRA STACK

TEST INFRA STACK

SERVER IMAGE: Build and test

serverspec, inspec, etc.
TEST JDK COOKBOOK

TEST TOMCAT COOKBOOK

BUILD OUR-APP

TEST OUR-APP

TEST ALL

QA

PROD

TEST APP SERVER ROLE

BUILD APP SERVER AMI

VALIDATE INFRA STACK

TEST INFRA STACK

INFRASTRUCTURE STACK
Validate (e.g. syntax, provision non-application specific infrastructure elements

awspec, inspec, etc.
INTEGRATED STAGES:

Test the various components in combination (deployment environments)

- Test JDK Cookbook
- Test Tomcat Cookbook
- Test App Server Role
- Build App Server AMI
- Validate Infra Stack
- Test Infra Stack
- Selenium, etc.
Recommendations

Those who write the code write the tests
Avoid reflexive testing (AKA testing your tools)
Test where there is risk, complexity
Test contracts
Splitting stacks into microstacks
Monolithic stack
Coordinating changes

Team A could make a change that breaks things for team B
Divide infrastructure into multiple, independently changeable stacks
Each stack has its own pipeline to deliver changes
Strategies for drawing boundaries
Draw boundaries to minimize changes that cross stacks
Infrastructure code boundaries ≠ network boundaries
Infrastructure code boundaries != network boundaries
Infrastructure code boundaries != network boundaries
Group things together when they tend to be changed together
Align team boundaries and change boundaries
Patterns for infrastructure stacks
Shared infrastructure components
Components shared across stacks

Base infrastructure module pipeline

Application stack pipelines
Challenges with shared components

Increases coupling
Shared infrastructure stacks
Shared stack instance

Multiple application infrastructure stacks

Shared network resource stack
Shared stack pipeline

Application infrastructure pipelines
Shared infrastructure stacks

The provider stack should not have knowledge of its consumers

Becomes a bottleneck as the number of consumers grows

Should be kept simple, minimal

Split into smaller stacks to avoid monoliths
Shared nothing stacks
Each stack is self-contained
Each stack is independently releasable
Techniques for integrating stacks
Example of integrated stacks

```python
vpc_id = "vpc-abcde"
private_subnets = ["sn-12345", "sn-54321"]
```
Integration at apply-time

```
vpc_id = "vpc-abcde"
private_subnets = ["sn-12345", "sn-54321"]
```
Integration by resource discovery

data "aws_vpc" "vpc" {
  filter {
    name = "tag:Environment"
    values = ["${var.env}" ]
  }
}

data "aws_subnet" "private_subnet" {
  filter {
    name = "tag:Environment"
    values = ["${var.env}" ]
  }
}
Integration by configuration registry

```
envs/qa/vpc_id       vpc-abcde
envs/qa/private_subnets "sn-12345", "sn-54321"
```
Integration by remote state
Recommendations

Stack integration points are contracts
Follow good practice for API design when making changes to contracts
Use fitness functions (automated tests) to ensure the integrity of contracts
kief@thoughtworks.com

Infrastructure Technologist@

ThoughtWorks

Twitter: @kief

Book: http://oreil.ly/1JKIBVe

Site: http://infrastructure-as-code.com