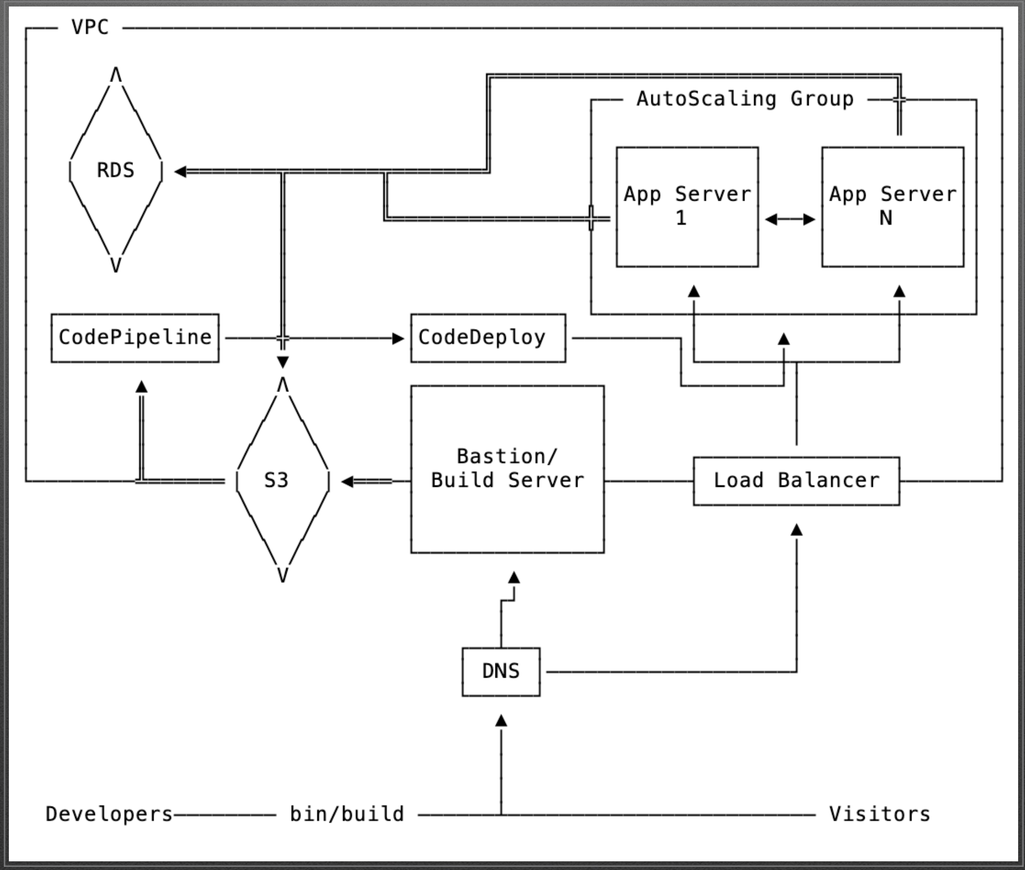
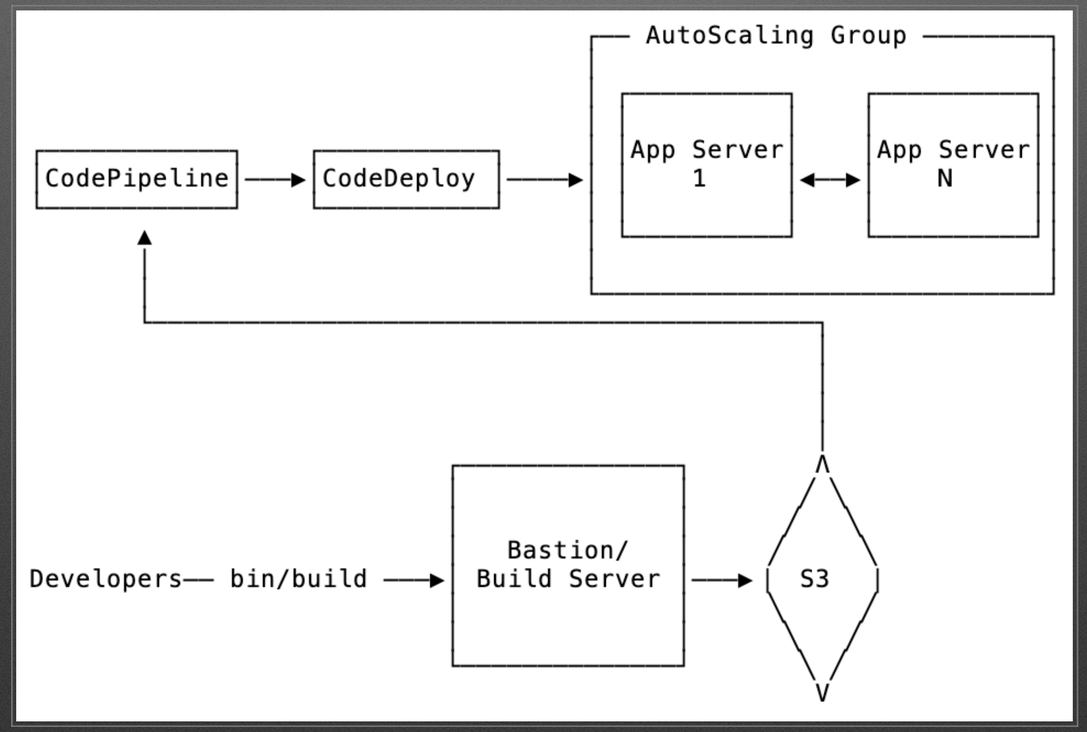
# A Deployment Pipeline

Elixir in the Jungle

# More Architecture



# Our Pipeline



# Review:  The Build

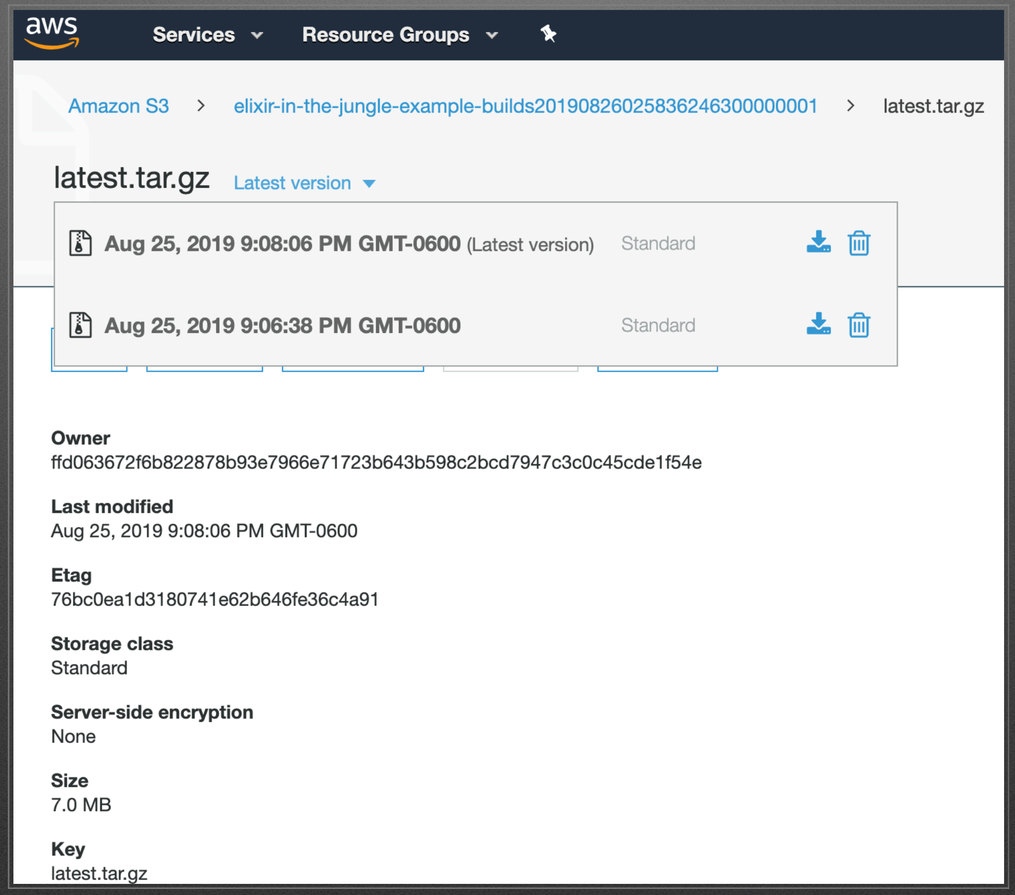
```
ssh -T $1 <<-SSHCMDS
    mkdir -p builds &&
    tar -C builds -xf $_archive &&
    cd builds &&
    mix local.rebar --force &&
    mix local.hex --force &&
    mix deps.get --only prod &&
    MIX_ENV=prod mix compile &&
    cd assets &&
    npm install &&
    npm run deploy &&
    cd .. &&
    MIX_ENV=prod mix phx.digest &&
    MIX_ENV=prod mix release --overwrite --path ./release &&
    cp -r ./scripts ./release/ &&
    cp ./appspec.yml ./release &&
    tar -C release . -zcf latest.tar.gz &&
    publish_release ./latest.tar.gz latest.tar.gz &&
    cd .. &&
    rm $_archive &&
    rm -rf builds
SSHCMDS
```

# `bin/build`

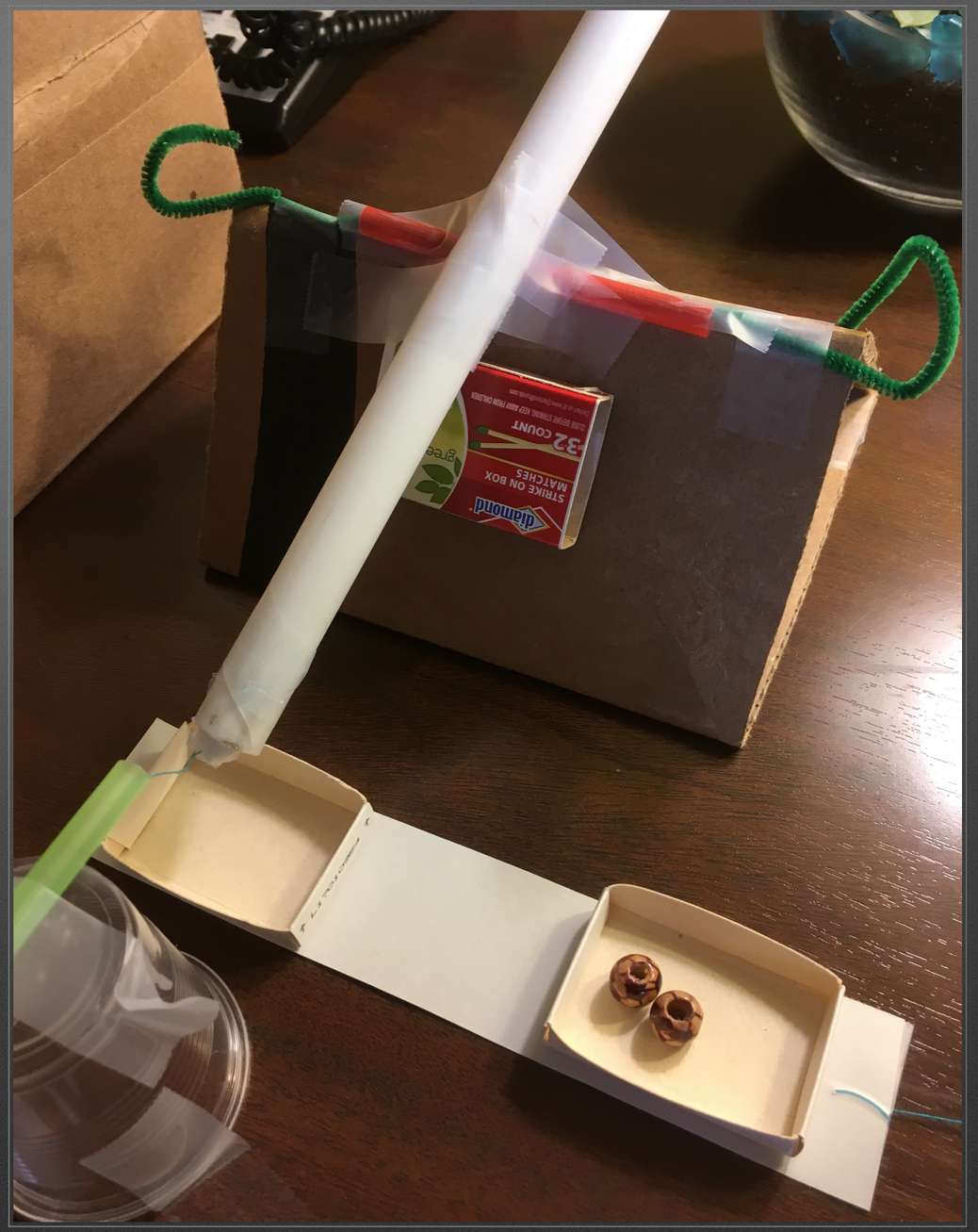
```
resource "aws_s3_bucket" "build_bucket" {
  bucket = "elixir-in-the-jungle-${terraform.workspace}-builds"
  acl = "private"

  tags = {
    Name = "elixir-in-the-jungle"
    Environment = "${terraform.workspace}"
  }

  versioning { enabled = true }

  lifecycle_rule {
    enabled = true

    noncurrent_version_expiration { days = 30 }
  }
}

resource "aws_s3_bucket_public_access_block" "build_bucket" {
  bucket = "${aws_s3_bucket.build_bucket.id}"

  ignore_public_acls = true
  restrict_public_buckets = true
  block_public_acls = true
  block_public_policy = true
}
```

# S3 Config

# Build Versions



# CodeDeploy



# Service:  CodeDeploy

- A service for automating software deployments
- Platforms:  EC2, ECS, Lambda, and more…
- Traffic control strategies:  Canary, Linear, and All-at-once
- Deployment types:  In-place, Blue/green, and more…
- Lifecycle events:  BeforeInstall, Install, AfterInstall, ApplicationStart, ValidateService, BeforeAllowTraffic, and more…

```
resource "aws_autoscaling_group" "app_server" {
  count = "${var.inital_asg ? 1 : 0}"
  name_prefix = "elixir-in-the-jungle-${terraform.workspace}-app-servers"

  desired_capacity = 1
  max_size = 2
  min_size = 1

  vpc_zone_identifier = ["${data.terraform_remote_state.vpc.outputs.private_subnet_id}"]

  load_balancers = ["${aws_elb.app_servers.name}"]
  health_check_type = "ELB"
  launch_configuration = "${aws_launch_configuration.app_server.name}"

  tags = [
    {
      key = "Name"
      value = "elixir-in-the-jungle"
      propagate_at_launch = true
    },
    {
      key = "Environment"
      value = "${terraform.workspace}"
      propagate_at_launch = true
    }
  ]
}
```

# Auto Scaling Group

k

```
resource "aws_autoscaling_group" "app_server" {
  count = "${var.inital_asg ? 1 : 0}"
  name_prefix = "elixir-in-the-jungle-${terraform.workspace}-app-servers"

  desired_capacity = 1
  max_size = 2
  min_size = 1

  vpc_zone_identifier = ["${data.terraform_remote_state.vpc.outputs.private_subnet_id}"]

  load_balancers = ["${aws_elb.app_servers.name}"]
  health_check_type = "ELB"
  launch_configuration = "${aws_launch_configuration.app_server.name}"

  tags = [
    {
      key = "Name"
      value = "elixir-in-the-jungle"
      propagate_at_launch = true
    },
    {
      key = "Environment"
      value = "${terraform.workspace}"
      propagate_at_launch = true
    }
  ]
}
```

# Auto Scaling Group

k

```
resource "aws_launch_configuration" "app_server" {
  name = "elixir-in-the-jungle-${terraform.workspace}-app-servers"

  iam_instance_profile = "${aws_iam_instance_profile.s3_app_access_profile.name}"

  instance_type = "${lookup(var.instance_size, terraform.workspace)}"
  image_id = "${data.aws_ami.ubuntu.id}"

  security_groups = ["${aws_security_group.app_servers.id}"]

  user_data = "${data.template_file.cloud_init.rendered}"
}
```

# Launch Configuration

WARNING:  Terraform can't "edit" this!

```
resource "aws_codedeploy_deployment_group" "app_servers" {
  app_name = "${aws_codedeploy_app.app_servers.name}"
  deployment_group_name = "elixir-in-the-jungle-${terraform.workspace}-app-servers"
  service_role_arn = "${aws_iam_role.app_servers_codedeploy.arn}"

  autoscaling_groups = "${aws_autoscaling_group.app_server.*.name}"
  load_balancer_info { elb_info { name = "${aws_elb.app_servers.name}" } }

  deployment_style {
    deployment_option = "WITH_TRAFFIC_CONTROL"
    deployment_type   = "BLUE_GREEN"
  }

  auto_rollback_configuration {
    enabled = true
    events  = ["DEPLOYMENT_FAILURE"]
  }

  blue_green_deployment_config {
    deployment_ready_option { action_on_timeout = "CONTINUE_DEPLOYMENT" }
    green_fleet_provisioning_option { action = "COPY_AUTO_SCALING_GROUP" }

    terminate_blue_instances_on_deployment_success {
      action = "TERMINATE"
      termination_wait_time_in_minutes = 5
    }
  }

  lifecycle { ignore_changes = [ "autoscaling_groups" ] }
}
```

# Deployment

# Health Checks

```
resource "aws_elb" "app_servers" {
  name = "elixir-in-the-jungle-${terraform.workspace}"

  # …

  # health_check {
  #   healthy_threshold = 2
  #   unhealthy_threshold = 2
  #   timeout = 3
  #   target = "HTTP:4000/ping"
  #   interval = 5
  # }

  health_check {
    healthy_threshold = 2
    unhealthy_threshold = 2
    timeout = 3
    target = "TCP:80"
    interval = 5
  }
}
```

# Health Check Configuration

```elixir
defmodule WebAppWeb.Endpoint do
  use Phoenix.Endpoint, otp_app: :web_app

  # …

  plug(:respond_to_ping, "/ping")

  plug WebAppWeb.Router

  # Here is where you should  test if the app can communicate with
  # the database other external services before going into service
  defp respond_to_ping(%{halted: true} = conn, _), do: conn

  defp respond_to_ping(%{request_path: path} = conn, path) do
    conn
    |> Plug.Conn.put_resp_header("content-type", "text/html")
    |> Plug.Conn.send_resp(200, "ok")
    |> Plug.Conn.halt()
  end

  defp respond_to_ping(conn, _), do: conn
end
```

# Custom Health Check Code

# Lifecycle Events

```yaml
hooks:
  AfterInstall:
    - location: scripts/setpermissions
      timeout: 30
      runas: app
    - location: scripts/getenv
      timeout: 30
      runas: root
  ApplicationStart:
    - location: scripts/start
      timeout: 30
      runas: app
  BeforeAllowTraffic:
    - location: scripts/findpeers
      timeout: 30
      runas: app
```

# Lifecycle Event Hooks

```
#!/usr/bin/env bash

set -e

sleep 10

/home/app/web_app/bin/web_app rpc "WebApp.ReleaseTasks.find_peers()"
```

```
defmodule WebApp.ReleaseTasks do
  def find_peers() do
    System.cmd("find_peers", [])
    |> case do
      {peers, 0} ->
        peers
        |> String.split("\n", trim: true)
        |> Enum.each(fn host -> Node.ping(:"web_app@#{host}") end)

      _ ->
        IO.puts("no hosts found")
    end
  end
end
```

# Connecting Nodes

```
#!/usr/bin/env bash

set -e

aws s3 cp s3://$BUILD_BUCKET/env /etc/environment --region us-east-2

_INTERNAL_IP=$(curl -s http://169.254.169.254/latest/meta-data/local-ipv4)

echo "" >> /etc/environment
echo "INTERNAL_IP=$_INTERNAL_IP" >> /etc/environment
echo "" >> /etc/environment
echo "RELEASE_NAME=web_app@$_INTERNAL_IP" >> /etc/environment
```

# Setting Environment Variables

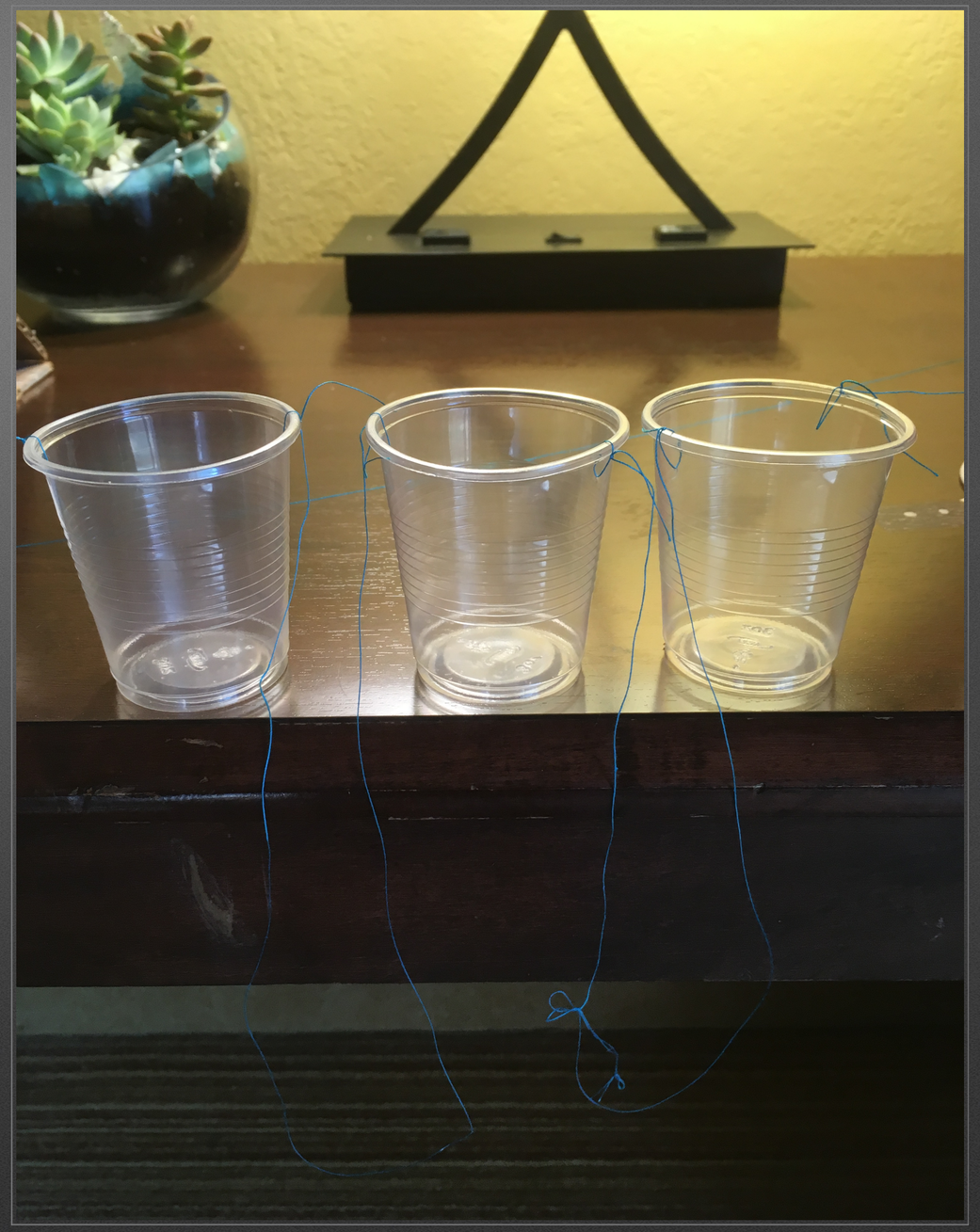
```
[Unit]
Description=Elixir In The Jungle
After=network.target

[Service]
Type=simple
User=app
Group=app
WorkingDirectory=/home/app/web_app
ExecStart=/home/app/web_app/bin/web_app start
Restart=on-failure
RestartSec=5
Environment=LANG=en_US.UTF-8
EnvironmentFile=/etc/environment
SyslogIdentifier=webapp
RemainAfterExit=no

[Install]
WantedBy=multi-user.target
```

# systemd

# A Manual Deploy

Demo

# CodePipeline



# Service:  CodePipeline

- A toolkit for automating release pipelines
- Support several stages:  Source, Build, Test, Deploy, and more…
- Each stage can integrate with multiple other services

# Our Stages:  Source and Deploy

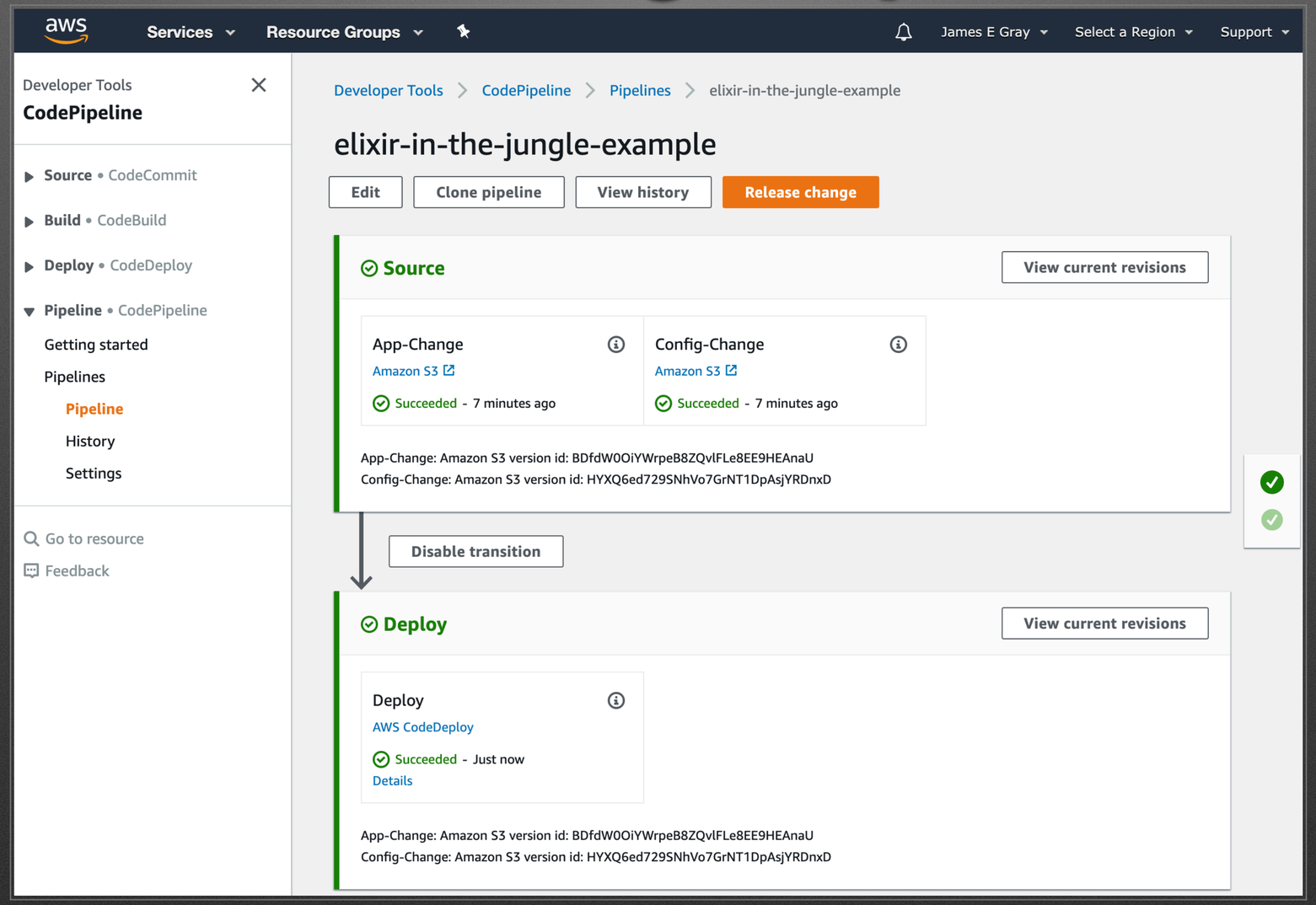
# The Source Stage

- Fed by an S3 bucket
- Watches for changes to release or environment files
- Hands a release forward to the Deploy stage

# The Deploy Stage

- Fed by the Source stage
- Hands the release to CodeDeploy for processing

# A Two Stage Pipeline



# Exercise:  Implement Our Two Stage Pipeline

- Add a `code_deploy.tf` to the provided Terraform files
- Tell CodePipeline to watch S3 for our release or environment files to change
- Handoff the deploy to CodeDeploy