2023 INTERNET2 TECHNOLOGY exchange

Control Chaos with IaC & Automation

Josh Whitlock
Sr. Cloud Engineer, University of California, Office of the President (UCOP)

1

About Us

UNIVERSITY OF CALIFORNIA Office of the President

Josh Whitlock
Sr. Cloud Engineer

- Systemwide infrastructure services
- Local infrastructure services
- > 50 cloud accounts

10 campuses

You've likely heard of our campuses before — each one of them ranks among the best in the world.

From UC Davis to UC San Diego, nine of our campuses house undergraduate and graduate education. UC San Francisco focuses on health education and it only offers graduate degrees.

6 academic health centers

Our six academic health centers have made some of California's biggest breakthroughs in health.

With top-ranked specialized care, support for clinical teaching programs, this is the home for spectacular innovation.

3 national laboratories

We help oversee three national labs for the US Department of Energy.

It's where cutting-edge sciences meet the curiosity and vibrancy of UC life. Since 1943, we've had 13 Nobel laureates pass through these doors.

2

About Us (cont'd)

UNIVERSITY OF CALIFORNIA Retirement At Your Service

The University of California Retirement Plan

- ~130,000 active members
- ~ 84,000 benefit payees including members, survivors and beneficiaries
- ~ $257M monthly payroll

REDWOOD

Redwood Pension Administration System

- 24x7 access by RASC staff, campus community partners and UCRP members via self-service
- ~150 interfaces and integrations

Where we thought we were starting from

Requirements

- The analysis and requirements gathering was completed by a previous consulting partner. They had been very thorough.
- System architecture, software requirements, interfaces already documented.
- We just had to get specific install instructions from the vendor.

Scope

- Migrate pension system from vendor hosted solution in Azure to AWS. Build out a defined set of identical environments to support the SDLC plans and deliver to users for testing.
- Scope was already defined and known for the most part. We just needed to start on builds.

Timeline

- An aggressive but reasonable timeline.
- No changes to functionality as part of migration, lift and shift.

#1 Challenge: Starting with false assumptions

Requirements

- Many hours of discussions and Q&A sessions with vendor and user community to get "real" requirements.
- Expecting thorough 600+ page documentation like most ERP install guides. Received a 12-page word document mostly with screen prints.
- *New requirement: gain user's trust and confidence.*

Scope

- Scope constantly changing. Basics like how many testing environments, what software was needed, what was included in each environment took months to nail down.
- Constantly in discovery mode (new databases to migrate, new software to install).

Timeline

- An aggressive but reasonable timeline. Required a major transformation for build timelines. We had to reduce months to weeks or days. Delivery had to be as close to flawless as possible.
- Lift and shift except if changes improved efficiency, delivery and reduced overall scope.
- "Like-for-like" mantra used to table discussions. "Add to the roadmap" became common too.

Challenge #2: The impacts of an unknown project timeline

Started out with:

- Can we do this?
- How do we do this?
- How long will this take?
- What do you mean we found more?

Then it becomes

- When can you have it?
- Why is it taking so long?
- Will the new environment have all the defects we had before?

Then it becomes

- Can you build two at a time?
- Can you build it in less than two weeks?
- Are we on track, will we make the go live schedule?
- Since that went so fast and well, could we…..?

Infrastructure as Code (IaC)

7

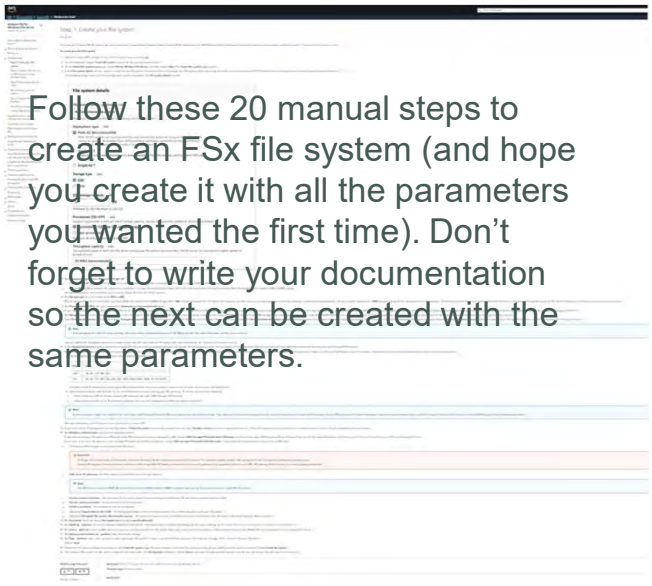
What is IaC and why do we rely on it?

- Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.
- Allows for consistent, repeatable deployments with approved configurations.
- Reduces risk. Use approved, secure configurations that are tested and validated one time but deployed infinitely.
- Adapts well to iterative, agile development cycles. Defects can be identified, resolved and deployed across a large number of systems in minutes.
- Deploy infrastructure at scale with extreme ease and speed.
- Increases operational efficiencies and reduces costs. Time is money. Utilize your most expensive resource, IT team members’ time, more efficiently.

What is Terraform?

- Terraform is an infrastructure-as-code software product created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON.
- Terraform codifies cloud APIs into relatively simple, declarative configuration files.

Terraform In Action



Follow these 20 manual steps to create an FSx file system (and hope you create it with all the parameters you wanted the first time). Don't forget to write your documentation so the next can be created with the same parameters.

Or use this simple Terraform block (and reuse it for the next, and the next, and the next)

```
resource "aws_fsx_windows_file_system" "example_fil01" {
  active_directory_id              = data.terraform_remote_state.msad.outputs.d
  aliases                          = [
                                       join(".", ["devfs2", data.terraform_remo
                                       "fileserver2.ucop.edu",
                                     ]
  automatic_backup_retention_days  = "7"
  copy_tags_to_backups             = true
  daily_automatic_backup_start_time = "02:00"
  deployment_type                  = "MULTI_AZ_1"
  preferred_subnet_id              = data.terraform_remote_state.vpc.outputs.pr
  security_group_ids               = [module.sg_fsx.id]
  storage_capacity                 = "650"
  subnet_ids                       = data.terraform_remote_state.vpc.outputs.pr
  tags                             = merge(tomap({ "Name" = "Example FSx" }), d
  throughput_capacity              = "64"
  weekly_maintenance_start_time    = "6:10:00"
}
```

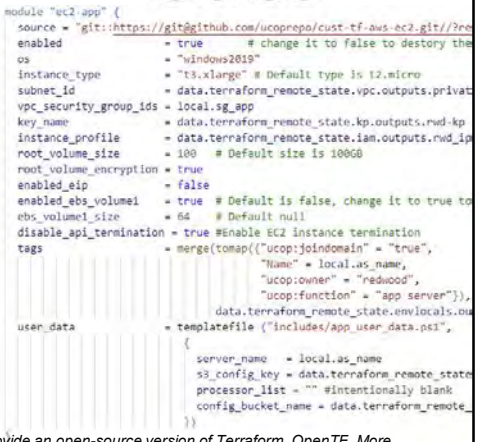
Leveraging IaC @ UCOP

- Any infrastructure we deploy instantly has our security and build standards applied, with no additional effort. Changes can quickly, easily and consistently be applied to our infrastructure resources. *
- We decided on a standard toolset. We chose Terraform as our code platform, Scalr for execution and state management and GitHub for version control. **
- We extended the TF language to create our own custom modules for standardized deployments.
- We regularly identify new resource changes and features to implement in our custom modules and iterate as our standards evolve. Modules along with our Terraform configuration files ar versioned in GitHub.

HashiCorp Terraform

scalr

GitHub



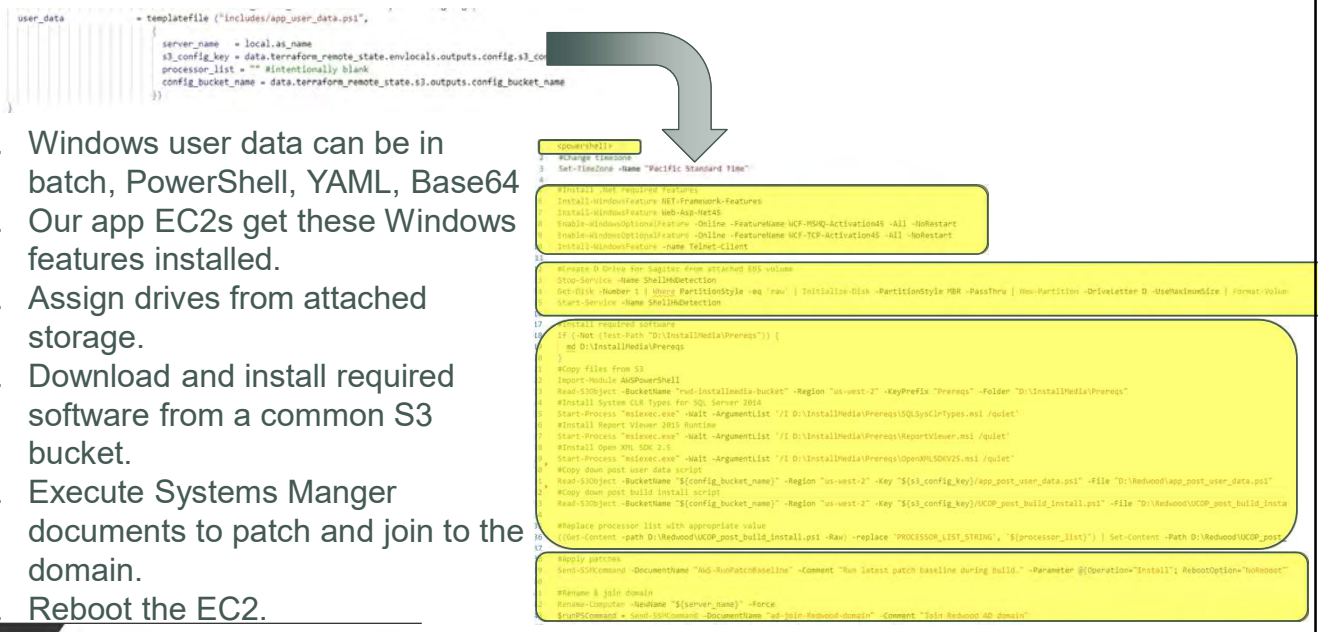
*Link to UCOP's 2022 Moving from Cloud Chaos to Standards presentation included on the end slide.

**There have been recent developments regarding HashiCorp's licensing of Terraform and an organized, community effort to provide an open-source version of Terraform, OpenTF. More information regarding OpenTF is available at https://opentf.org.

A Powerful Combination: IaC & Automation

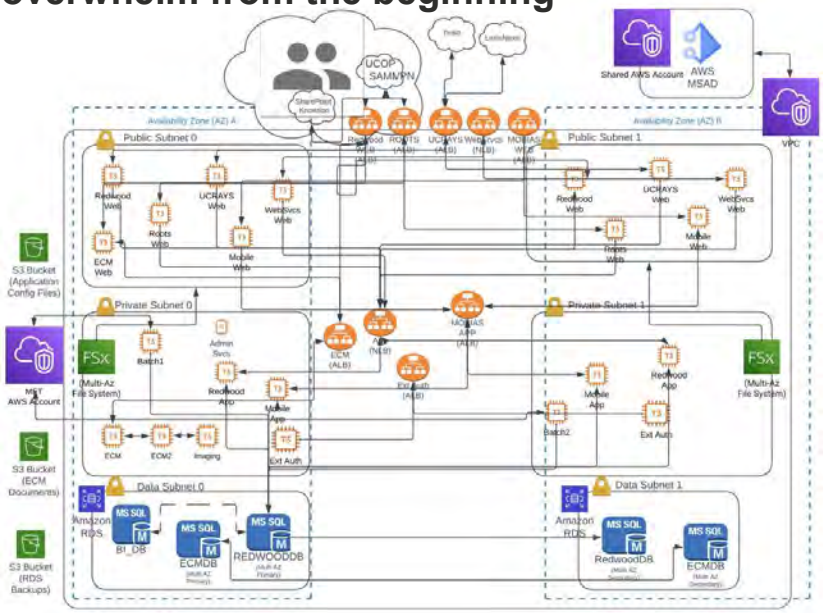
- Some infrastructure resource changes can automatically trigger automation (i.e. kick off PowerShell scripts, execute SSM documents, etc).
- Convert application pre-requisites and installs to native PowerShell scripts. Remove or reduce complex, error-prone manual installation steps while increasing consistency and quality of your delivery.
- Once coded, automations can be applied across infinite resources at time of deployment or via tags, resource groups or other metadata variables.
- Allows for faster deployments that can be reused across any number of environments and build combinations.

Automation in action

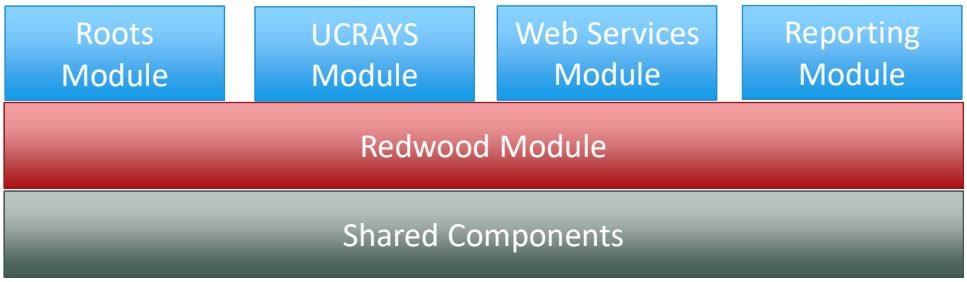


1. Windows user data can be in batch, PowerShell, YAML, Base64
2. Our app EC2s get these Windows features installed.
3. Assign drives from attached storage.
4. Download and install required software from a common S3 bucket.
5. Execute Systems Manger documents to patch and join to the domain.
6. Reboot the EC2.

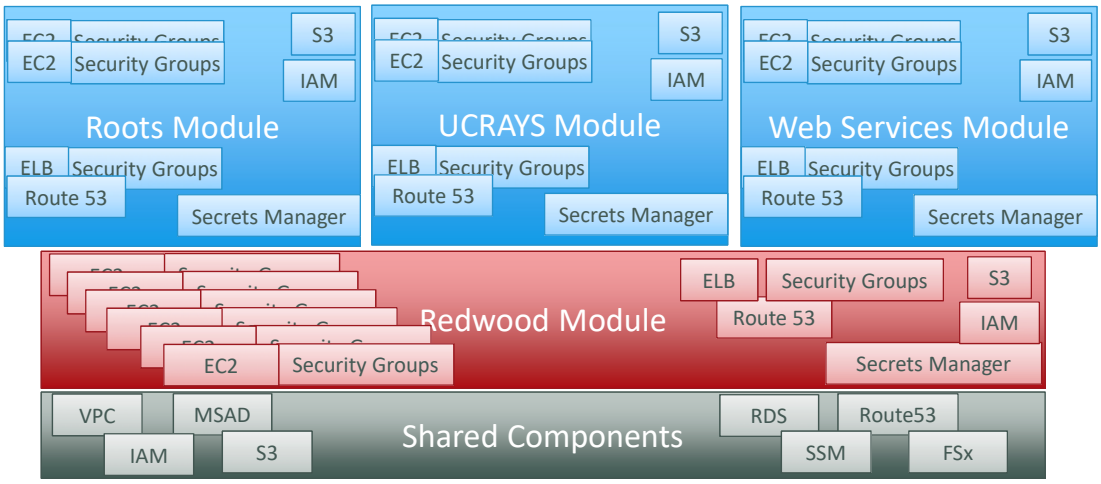
Chaos and overwhelm from the beginning



Taking a modular approach to design



Breaking each further into reusable blocks



Every PM's Dream

Shrinking Timelines Instead of Increasing Them

Leverage Reusable Code to Deliver Ahead of Schedule

- Our initial full environment build ~ 5 months

  Included many hours of requirements gathering, documentation, infrastructure revision and script development.

- Our second environment ~ 6 weeks

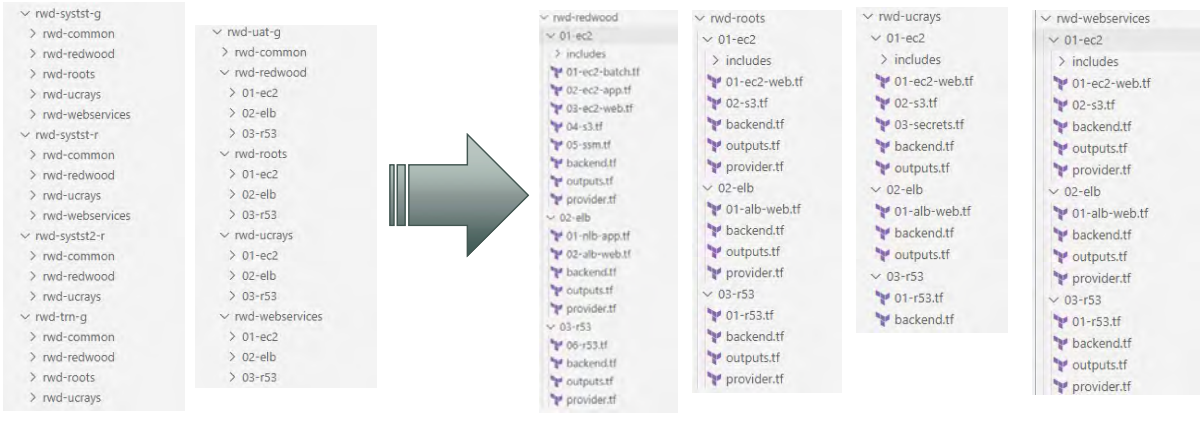
  No overlap in builds or resources. Included previously unknown requirements, software and features as well as defect fixes we were able to bring in from lessons learned in first environment.

- Subsequent environments = 4-10 days

  Each build is iterative. As defects are identified in any environment, we resolve and implement the fix into our build process to prevent regression issues. We apply fixes easily across all environments within our code.

Building 10 environments using copy and paste

Utilizing a standardized structure, we were able to build our environments in a "cafeteria tray" style. Choose the blocks to use based on the requirements we were provided with. If they changed after, it was just as easy to add another block.



21

The Law of Good, Fast and Cheap

- You can have it *good and fast*, **but it won't be cheap**.
- You can have it *good and cheap*, **but it won't be fast**.
- You can have it *cheap and fast*, **but it won't be good**.

We knew migrating an enterprise application with no documentation and an ever-evolving list of requirements would not be cheap. There was also a high potential for inconsistencies and quality issues that will undermine stakeholder confidence (not good). Based on the first build, we had our doubts if it would ever be fast.

Which two could we pick?

By investing heavily upfront in analysis and creating a very solid technical foundation, we were able to successfully deliver on both good and fast. We used a component based approach utilizing IaC and automation to deliver high quality, consistent environments for our users.

22

Future improvements for faster and easier deployments

As you can expect, we are not stopping here. But to meet our timeline, we must draw a line on what to include. There are numerous areas will evaluate and improve upon post go-live.

- Further automations – identify weaknesses and remaining manual steps. Put scripted automation in place where possible. Train staff to look for these opportunities.
- Evaluate use of custom images and containers.
- Develop a blue/green deployment strategy to reduce risk and downtimes for patching/upgrades.
- Evaluate and replace components with additional AWS managed services.

UCOP @ Technology Exchange

Join us for our 2023 Technology Exchange presentations by UCOP team members:

- **Moving from VM to Cloud Native Containers with Khalid Ahmadzai, Tuesday 11:20am-12:10pm**
- **Cloud Security By Default with Matthew Stout and George Holbert, Thursday 10:20am-11:10am**
- **Control Chaos with IaC & Automation with Josh Whitlock, Thursday 1:40pm-2:30pm**

2022 Technology Exchange presentation by UCOP's own Khalid Ahmadzai, Kari Robertson, Matt Stout

Moving from Cloud Chaos to Standards:
https://internet2.edu/wp-content/uploads/2022/12/techex22-Cloud-MovingfromCloudChaostoStandards-AhmadzaiStoutRobertson.pdf

25