Infrastructure as Code

Deployment Recovery and Automation Technology (DRAT)

John Klein

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0315

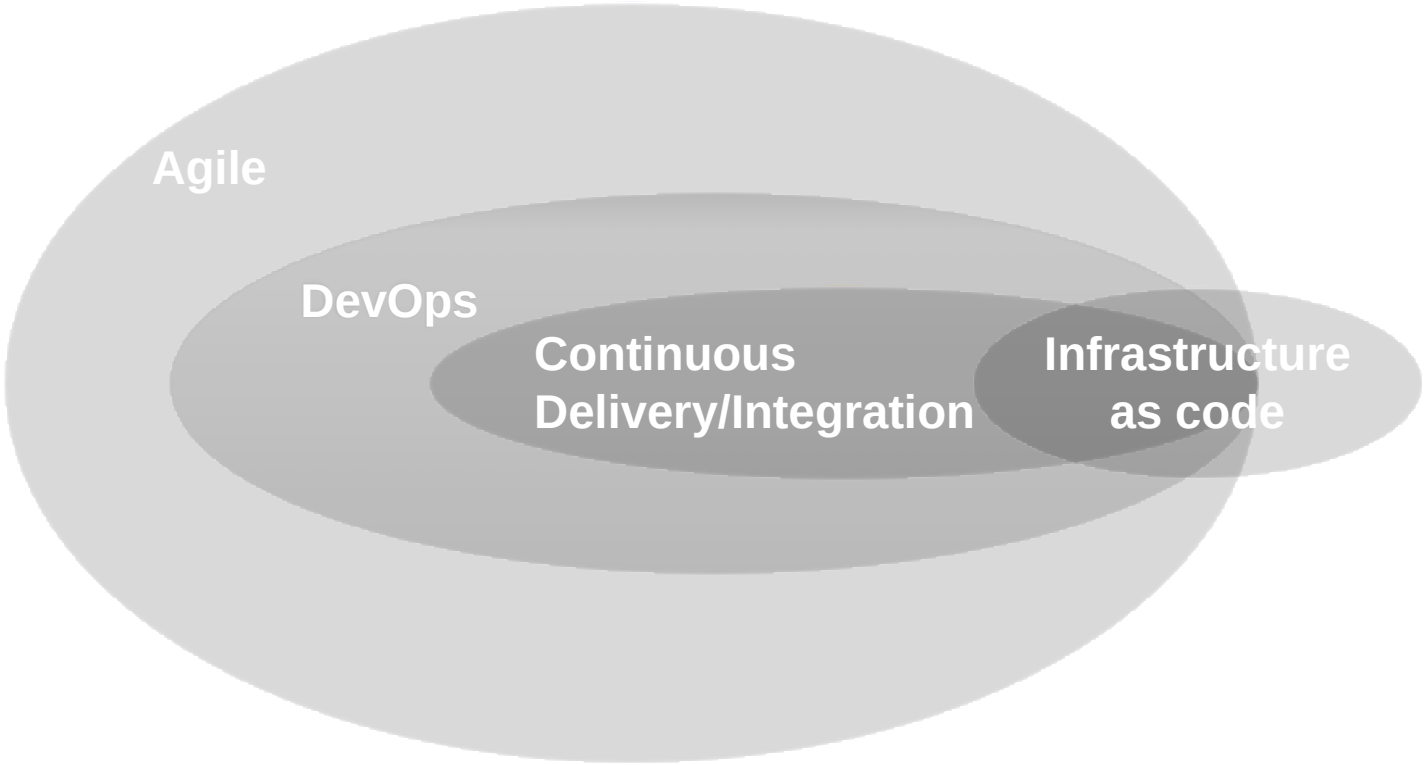
Outline

Background

Problem

Approach

Infrastructure as Code* – Benefits Beyond Agile



Today
- **Automated Deployment**
- **Immutable Infrastructure**
- **Versioning and Rollback**
- **Environment Parity**

Future
- **Portability across IaaS**
- **Assurance evidence**
- **Moving Target Defense**

***Process and technology to manage and provision computers and networks (physical and/or virtual) through scripts**

DoD Problem

Software Engineering Centers (SECs), Software Maintenance Groups (SMGs), and other sustainment organizations want to realize the benefits of infrastructure as code

They must first recover the technical baseline for the deployment

- Infrastructure as code doesn't exist for legacy systems, or no government data rights to contractor deployment scripts
- Often the only authoritative artifact is an instance of the running system

Can the deployment structure be automatically recovered from an instance of the running system?

Problem arises from acquisition context – SEI is uniquely positioned to solve it

Why is it hard for DoD to adopt Infrastructure as Code practices?

Deployment scripts are manually coded

- Requires specialized skills and knowledge – infrastructure design, deployment tools, *and* internal design of the system
- Complex and error-prone – usually, errors are detected quickly and rolled back
  - Not always, e.g., AWS US-EAST-1 Region outage in February 2017

Recover the baseline = inspect every node in the deployment (10s-100s)

Tempo mismatches

- Deployment can be changing frequently during development and active sustainment – infrastructure as code needs to match this tempo
- IaC tools are open source and rapidly evolving, infrastructure code needs to evolve to stay current even if there is no ongoing active sustainment

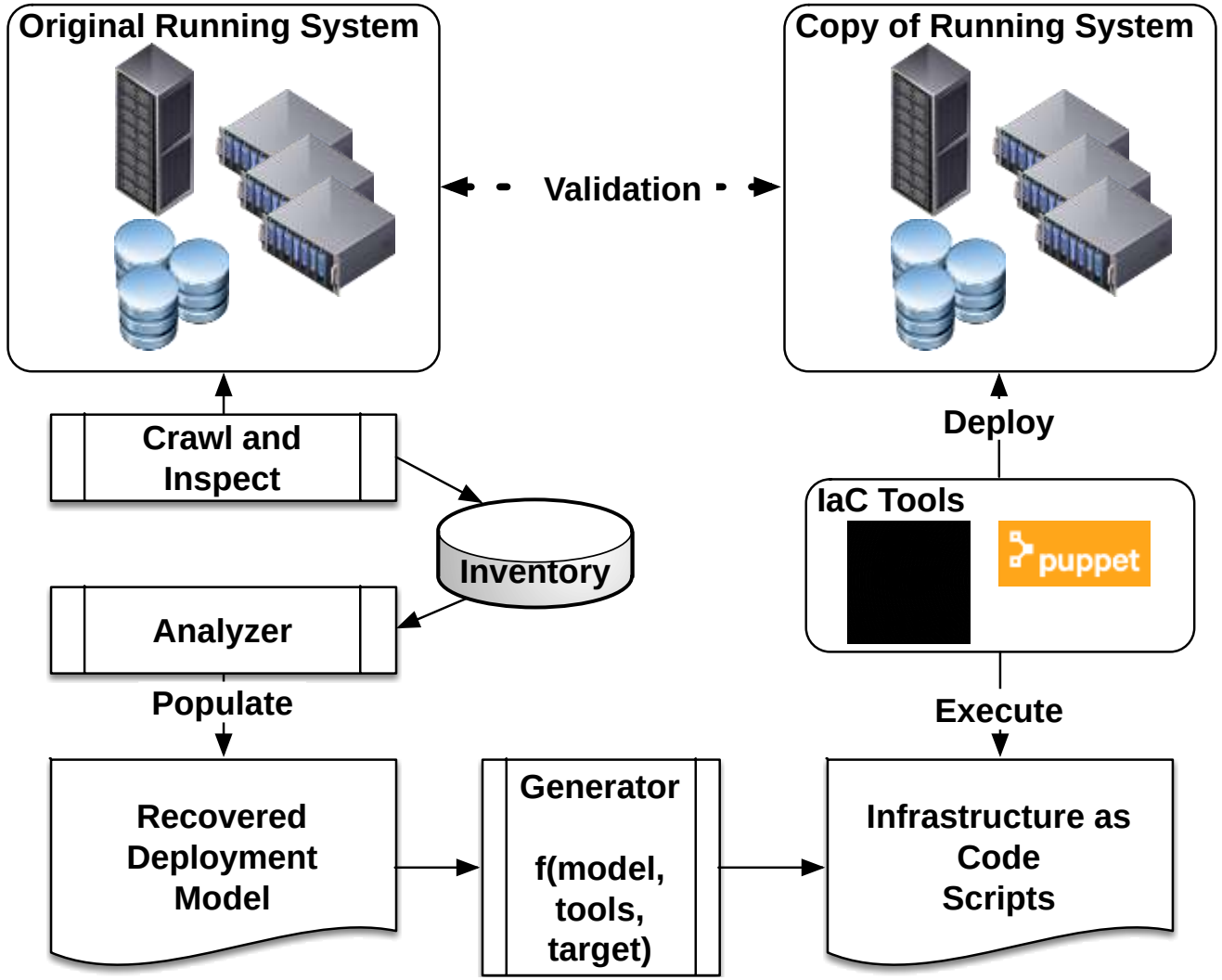
Result: "IaC is too hard/takes too long. We'll use traditional operations practices, and deal with this later."

Solution Approach - Technology

Automatically recover a deployment model from running system and generate IaC scripts from model

Model-based deployment enables automation:

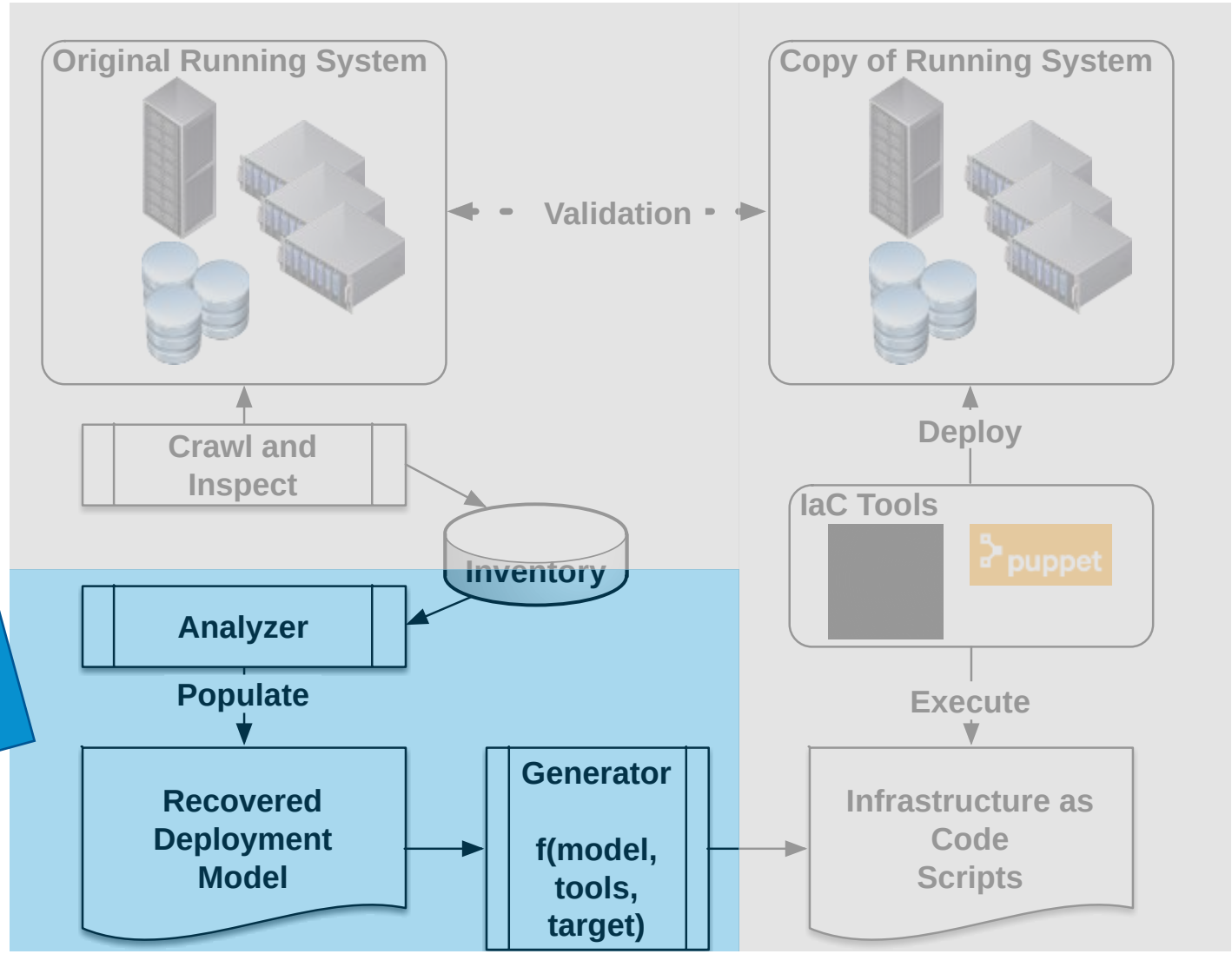
- Port scripts to new tools or IaaS
- Analyzing model against design rules
- Transform model (moving target defense)



Solution - Technology

Research Focus

- Design deployment model
- Analyze inventory and populate model
- Generate IaC scripts from the model



Analysis “Principles”

1. We care about files that can execute.
   - OS
   - Started by service manager (e.g., initd, serviced)
   - User runs command in shell
2. Files that execute use other types of files:
   - Management scripts (start, stop, restart, …)
   - Configuration
   - Content (e.g., content delivered by a web server)
3. Files originate in a limited number of ways
   - Part of OS installation
   - Part of an installed package and unmodified
   - Part of an installed package, modified (e.g., config file)
   - Executable that is not part of an installed package (e.g., copied onto the system)
4. Linux Filesystem Hierarchy Standard provides a laundry list of places to look for executables (in a well-behaved environment)

Analysis Approach

All files start as *Origin* = *UNKNOWN*. We will *mark* each file's origin as we discover it

First, take care of the easy stuff

- OS
- Installed packages, modified and unmodified files
- Stuff that you **don't** want to propagate (e.g., /proc, much of /var, …)

Apply heuristic rules

Make it easy to add rules

- Easily extensible framework (*Strategy Pattern*)
- Utility functions for common operations

Heuristic Rule Types – "What would an expert do?"

Package-specific rules, e.g.,

- Is Postgresql package installed? If so, do we start the service? If not, we're done.
- If we start the service, parse the configuration file to ID the data directory and mark that as "content"

User-installed services (not part of a package)

- RHEL/CentOS puts these in /etc/systemd/system/multi-user.target.wants
- Look there, ignore the usual suspects (e.g., sshd)
- For each service we find, parse the configuration file. Find references to files/directories, and mark that as user-installed.

User home directories - /home/*

- Parse shell initialization dotfiles, look for additions to $PATH, mark content from those directories.

Generation Approach

Targeting Ansible automation engine

- YAML-based script syntax
- Using “generic” automation engine features, could generate for other tools such as Chef

Provision VM, if needed.

Start from image for appropriate Linux distribution/version

Install packages

Apply patches for modified files.

Copy files that don’t come from installed packages.