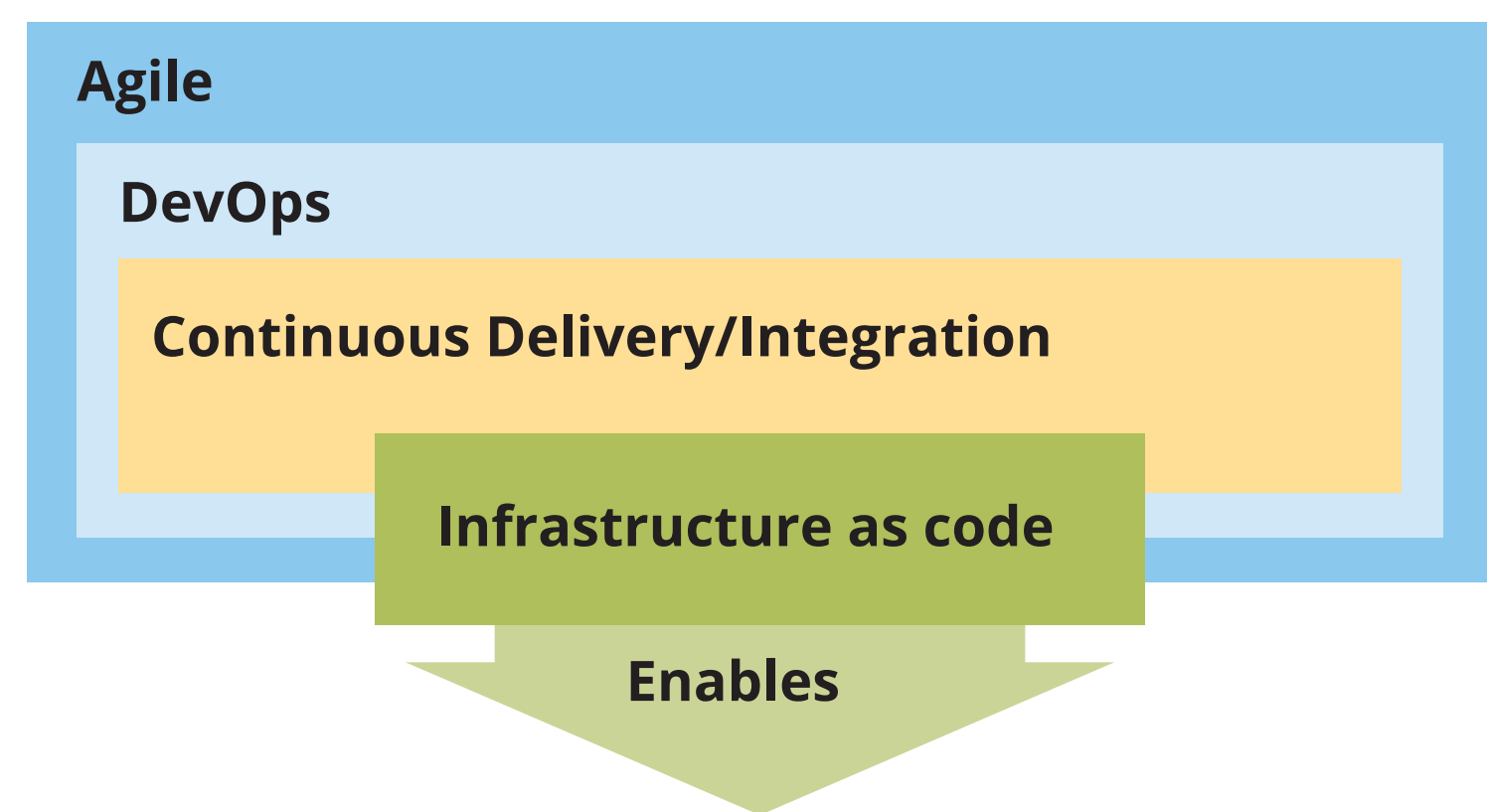


Infrastructure as Code

Feasibility of recovery of software deployment architecture

DoD sustainment organizations want to adopt agile practices and realize the benefits of DevOps and infrastructure as code (IaC). They must first recover the technical baseline for the software deployment. This project has prototyped technology to automatically recover the deployment baseline and create the needed IaC artifacts, with minimal manual intervention and no specialized knowledge about the design of the deployed system.

IaC is the process and technology to manage and provision computers and networks (physical or virtual) through scripts. IaC is a foundation of integrated development and operations (DevOps) that provides automated deployment to the integration environment and repeatability through immutable infrastructure, enables exploration and experimentation by providing environment versioning and rollback, and ensures parity of test and integration environments across locations and organizations. IaC is usually associated with Agile and DevOps, but it can provide benefits outside of Agile.



- | Today | Future |
|----------------------------|---------------------------|
| • Automated deployment | • Portability across IaaS |
| • Immutable infrastructure | • Assurance evidence |
| • Versioning and rollback | • Moving target defense |
| • Environment parity | |

Our approach has four elements. We **crawl** through an instance of the deployed system and inspect each node to create an inventory of software. Next, we **analyze** the inventory and “make sense of it” — identify which software is part of the operating system, which other packages are installed, and which is the application software. From this analysis, we populate a **deployment model** of the system. From the deployment model, we **generate** the scripts needed by the infrastructure as code tools, which execute the scripts to create a new deployment of the system



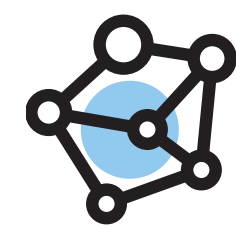
Crawl: Our crawler uses a novel approach to execute a script written in the Python programming language on the source system without installing additional software.



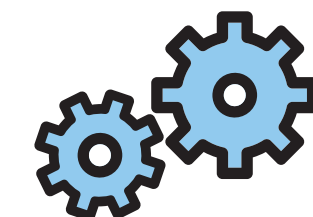
Analyze: We first determine the source repository for each installed package and associate files to installed packages. We then run a set of heuristic rules that uses file patterns to identify configuration files and pattern matching within configuration files to identify directories and files added to the system outside of installed packages.

Heuristics classify files by source. The heuristic rules infer identity and source of files that are not installed with a package, such as:

- Content served up by an installed web server
- Scripts or services delivered from an installed web container like nginx or Apache Tomcat
- Configuration and schema definition files for an installed database
- Standalone user services or applications

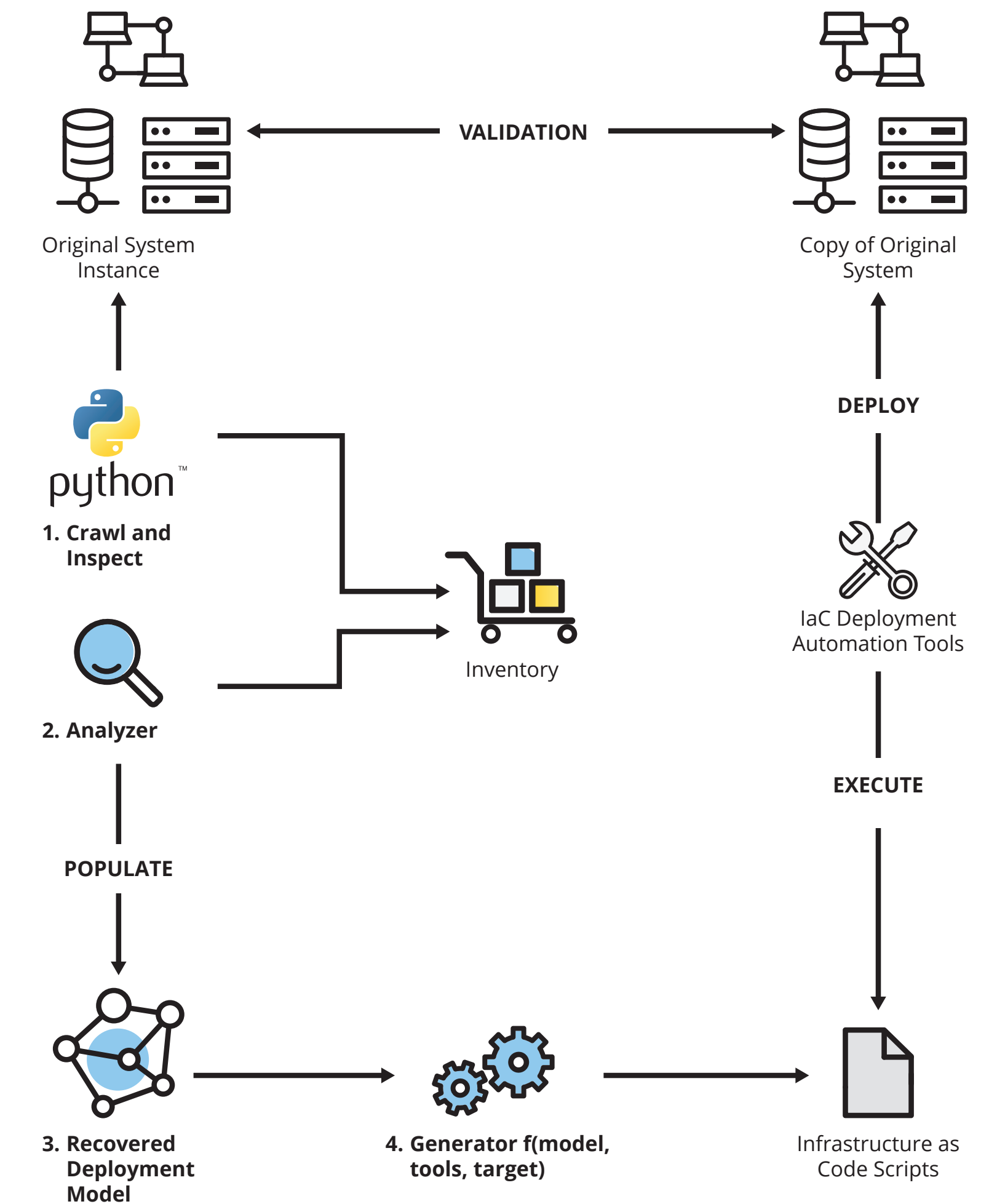


The **deployment model** is a relational schema that represents all of the facts and inferences.



Generate: Our prototype generates a set of scripts for the open source Ansible automation tool. The prototype can be extended to generate scripts for other tools.

Approach: Crawl, analyze, populate a model, and generate IaC artifacts.



Limitations and future work: Our approach is limited to Linux-based systems. We have demonstrated an initial set of heuristic rules covering a number of inference types and patterns, with extensibility to add new rules to broaden coverage.

Software sustainment organizations can use this tool to quickly understand a system, and create and run automated deployment scripts to enable exploration and evolution.

Infrastructure as Code

Copyright 2018 Carnegie Mellon University. All Rights Reserved.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-1132