Provenance Issues in Platform-as-a-Service Model of Cloud Computing

Devdatta Kulkarni
devdattakulkarni@gmail.com

PhD, University of Minnesota Minneapolis
Affiliations: Rackspace, UT Austin

Agenda

Define Provenance in PaaS

Discuss Provenance Issues in PaaS

Present mechanisms to address the issues

PaaS

Systems that allow application developers to deploy their applications to cloud infrastructure “easily”

Without having to worry about provisioning the infrastructure layer (servers, database, etc.)

Also known as application life-cycle management (ALM) systems

E.g. Heroku, Google App Engine, OpenShift, CloudFoundry, Solum

Provenance

Information about an entity that helps with understanding how that entity got to a particular state

The “entities” that we consider are

- The platform itself
- Applications deployed by the platform

Provenance Examples

When developing PaaS, what were the exact set of commands used to install a particular library/tool?

When a PaaS is deployed, what are the values of the configuration parameters for different services?

When an application is deployed by PaaS, what is the version of Docker used to build application containers?

PaaS and Provenance

PaaS manages complete life-cycle of an application

Provenance is important

- For PaaS developers and operators
  - To enable correct design and operation of the PaaS
- For application developers
  - To gain insights into application construction process
  - To gain confidence in the working of a PaaS

Solum - PaaS / ALM for OpenStack

- Supports deploying applications starting from the source code
- Custom Language Pack mechanism

  Java, Python, NodeJS, Ruby, …
- Applications are constructed as Docker containers
- Uses OpenStack services

  -Keystone for authentication

  -Glance and Swift to store container images for language runtimes, application container images, logs

  -Heat and Nova to deploy application containers
- Allows optionally running of tests

  Continuous integration for applications
- Is integrated with Github

  Application deployment can be triggered by Github webhooks

Agenda

Define Provenance in PaaS

Provenance Issues in PaaS
- **PaaS development**
- **PaaS building**
- PaaS deployment
- Applications on PaaS

Mechanisms to address the issues

PaaS development

Solum experience

- Several softwares required, such as Docker, Docker registry, Tomcat, Swift, Glance, Keystone, Nova
- Installation of a s/w before it can be used was a trial and error process

Typical command line contains lot of commands of different kinds

Navigational commands (cd, pusd), Listing/viewing commands (ls, less), editing commands (vi, emacs)

Provenance for PaaS development

- Once a s/w has been successfully installed, we don't want to repeat the process from start again
- Is it possible to *automatically* generate the list of commands required to install a particular s/w?
- We needed *provenance* of software installs
- This requirement is not confined to development of PaaS but arises in any development scenario that needs to install and use new s/w

PaaS building

Solum experience

- OpenStack services progress independently
- Changes in dependent services may cause Solum to stop building

We want to unblock Solum builds by pinning to an earlier commit of a service

Figuring out commit(s) breaking Solum in dependent services is a manual and tedious process

Provenance for PaaS builds

Realization -

Every time Solum is successfully built, maintain information about the commits of the dependent services used (similar to “Global restore points” of App-Bisect)

- This information will be useful when Solum builds fail in the future
- Finding culprit commit in a dependent service can start from the last known good commit with which Solum was successfully built
- We needed *provenance* of successful Solum builds

git bisect for using dependent services

Agenda

Define Provenance in PaaS

Provenance Issues in PaaS

Mechanisms to address the issues

- **PaaS development**
  - **Command List Provenance**
- **PaaS building**
  - **Commit tracking**
  - **Merge tracking**
- PaaS deployment
  - Configuration parameter tracking
- Applications on PaaS
  - Object model and API

Command List Provenance

Problem

Given shell history find the list of commands that represent provenance of a software's installation

- Solution outline
  - –Create candidate list of commands
  - –Try the candidate list in an automated manner
  - –Verify that the candidate list leads to software's installation (the software's provenance)
- Challenges
  - –How to deal with long shell histories?
  - –How to determine that a candidate list of commands represents the provenance of installing a software?

Command List Provenance

- To address long shell history
  - Define Anchor Point (AP) commands
  - These are commands which provide starting and stopping points within the shell history

  E.g.: apt-get update on Ubuntu

- To address automated trial and verification of command list
  - Use capabilities offered by *Docker*
  - Build a *Docker container* with candidate command list
  - Verify the list using a *verification script*

- Verification script

  Defines checks to verify that the software was correctly installed

Command List Provenance

- Feasibility study (initial experimentation)
  - Installed Docker and Tomcat on Host
  - Tried the command list provenance approach to find out provenance for both
- Verification Scripts
  - For Docker

    Check output of *"docker –v"* command
  - For Tomcat

    Check presence of *"webapps"* folder at a known location

Command List Provenance

Command histories

Tomcat

% :
% tomcat
% apt-get update
% apt-get install -y tomcat7
% curl localhost:8080

Docker

% apt-get upgrade
% apt-get update
% apt-get -y upgrade
% uname -r
% which wget
% wget -qO- https://get.docker.com/ | sh
% docker -v

Sample Observation

Observation

Docker build may fail if:

- Command is a not-existent command
  - Happened with *tomcat*
- Command is not installed on the container (any layer)
  - Happened with *curl*
    - » Was present on the host but its installation was not part of the candidate command list

Realization

Before including a command in *Dockerfile* run it on the host

- Include it only if it ran successfully on the host
  - Ruling out inclusion of *tomcat*
- Include installation command
  - Ensuring *curl* is installed on the container

Command list prov: Observations (2/3)

Observation

A command may need different flags to execute on container as compared to the host

- Happened with certificate checking by wget
- Had to introduce --no-check-certificate flag on container
  - wget -v --no-check-certificate https://get.docker.com/

Realization

Need to figure out appropriate set of flags to use when including the command in Dockerfile

Command list prov: Observations (3/3)

Other observations

- Piped commands on host may need to be split before including in Dockerfile
- Navigation commands (cd/pushd/popd) need to be combined with other commands when including in the RUN command in Dockerfile
- Files modified on the host can be copied into the container at appropriate location

Command list prov: Other Issues

How to handle external dependencies and their versions when the s/w was installed on the host?

One idea is to determine versions of external packages when the software was installed on the host

- Possible to find out using:
  - dpkg –l
  - pip freeze

Service dependency tracking

Problem

Given a failing Solum build how to determine which service and which commit of it is the cause of the build failure

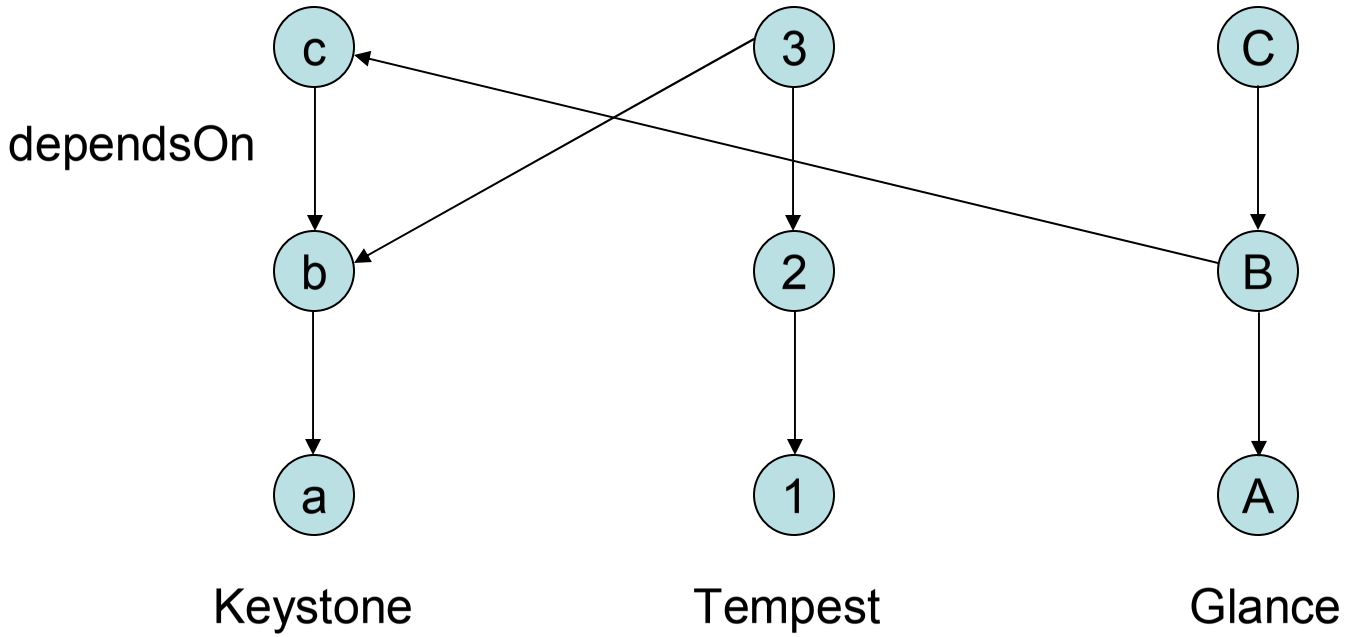
Solutions

- Track dependent services and their commits
- Track dependent services and their “merge-to-master” events

Commits tracking

- Track commits of dependent services that lead to successful Solum builds
  - Last successful build:
    - *<Keystone=a, Tempest=1, Glance=A>*
- Suppose Solum build fails and the current commits of dependent services are:
  - *<Keystone=c, Tempest=3, Glance=B>*
- How to determine which service and which commit is causing Solum build to fail?

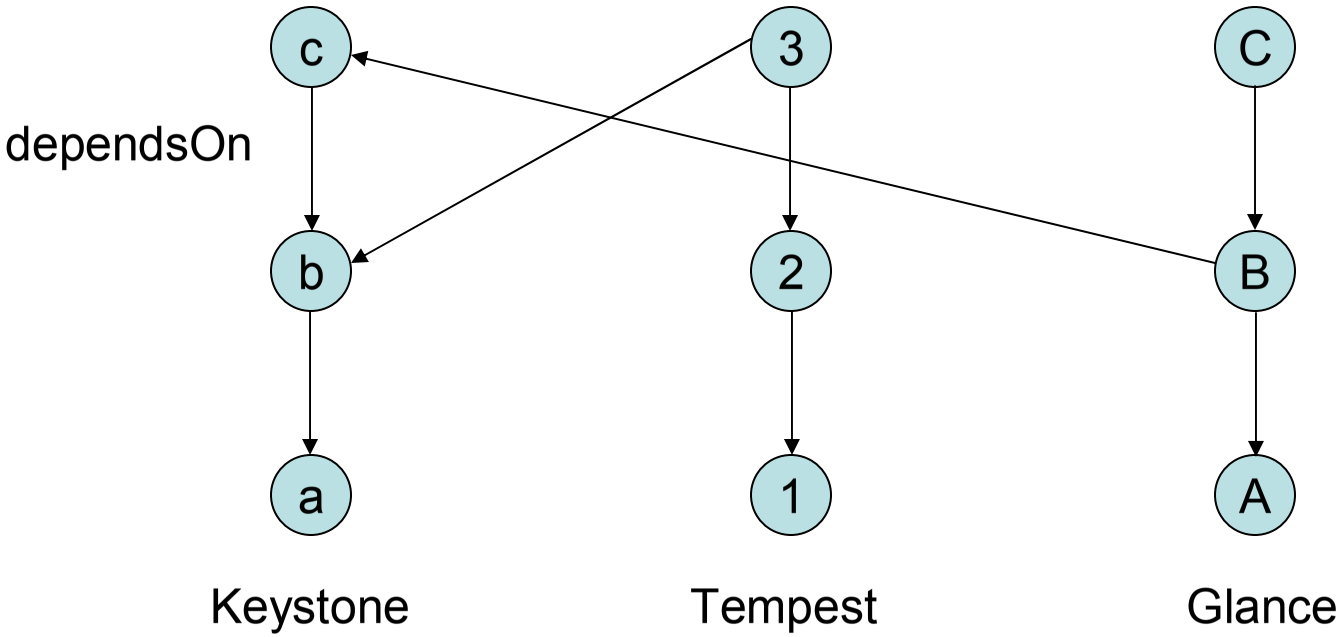
Commit dependency graph



Finding breaking commit

- Select a dependent service S
- Check the latest commit to see if Solum builds successfully
- If not, remove the commit from consideration
- Remove all the commits from other services that form a *transitive closure* of the *needed-by* graph (edges reversed from the dependency graph)

Commit dependency graph

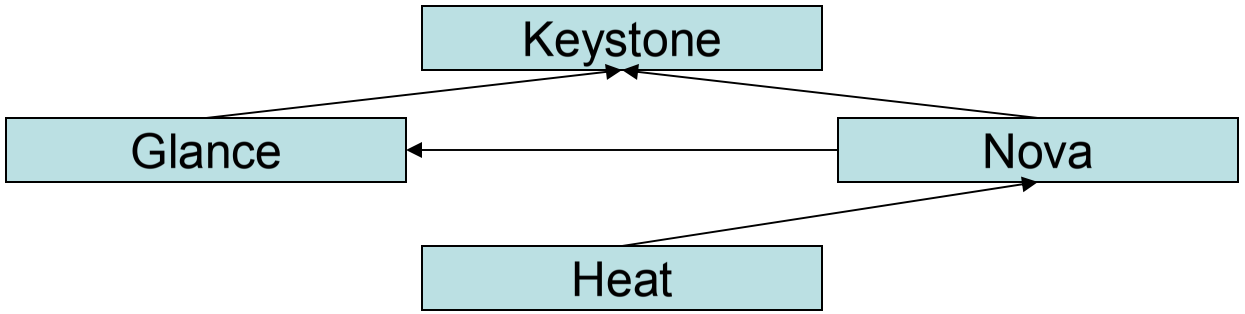


Candidate commit:<Keystone=c, Tempest=3, Glance=B>

Candidate commit:<Keystone=b, Tempest=3, Glance=A>

Questions

- How to build the dependency graph?
  - Possible if each service maintains provenance for its successful builds
  - Service dependencies form a DAG



- In which order dependent services should be tried?

Merge tracking

Idea

Whenever new code merges to master in dependent services, *proactively* check that it does not affect Solum

OpenStack CI Systems

- Zuul
  - Runs tests
  - Project definition
    - which tests to run
- Gerrit
  - Code gating and reviews

Merge tracking

- Enhance project definition in Zuul with a *Trigger* event and list of *using* projects

  Barbican:
  Triggers:
  OnMergeToMaster:
  Used_by:
  Solum, Murano, Mistral

- On merge-to-master add a “recheck nobug” comment to outstanding patches for *Used_by* projects in Gerrit
  - This comment triggers a CI run on the project

Questions

- What modifications are needed to Zuul to enable merge-to-master event generation?

- What if no outstanding patches are present for a project?
  - Should Zuul generate a patch?
  - What will be the nature of such a patch?

Provenance of PaaS deployment

Solum experience

- Each OpenStack service has large number of configuration parameters
- Not setting correct parameters in dependent services to appropriate values caused Solum to not deploy

Problem

Need to know which parameters in dependent services are critical for Solum deployment

Potential solution

Track parameters and their values for dependent services

Infrastructure Configuration tracking

Problem

On development version of OpenStack (devstack), config parameters and values of dependent services need to be tracked

Solution

Version control

Provenance of application on PaaS

- Which revision of application code used to create application container in a particular deployment?
- Which revisions of system libraries and software used for application construction, building, and deployment?
  - Revision of Docker used
  - Revision of Heat used
  - Revision of Glance/Swift used

Application Provenance API

Problem

Need to maintain information about every application build and deployment

- Application-level information
  - Source code commit
  - Test and run command used
- Infrastructure-level information
  - Version of Docker used
  - Version of Heat used

Solution

- Mechanisms to track this information
- API to extract it

PaaS development

- Other aspects
  - Online tutorials used
  - Code samples used
  - Stackoverflow links referred to
- Tracking provenance of platform code development
  - Not considered

Conclusion

- Presented issues arising in developing and building PaaS
- Argued that *provenance* can be used to address these issues
  - Identified the needed provenance information
  - Presented mechanisms to collect and use this information
- More details available in the paper