SID

Software and Infrastructure Deployment

Agenda

1. Concepts
   - Glossary
   - Workflows
   - Versioning
2. Architecture
   - Projects management
   - Workspaces
   - Templating
   - Project configurations
   - Deployment
3. About the API
4. Workshop

CONCEPTS
GLOSSARY

escaux

GLOSSARY

Terms mapping at Escaux:

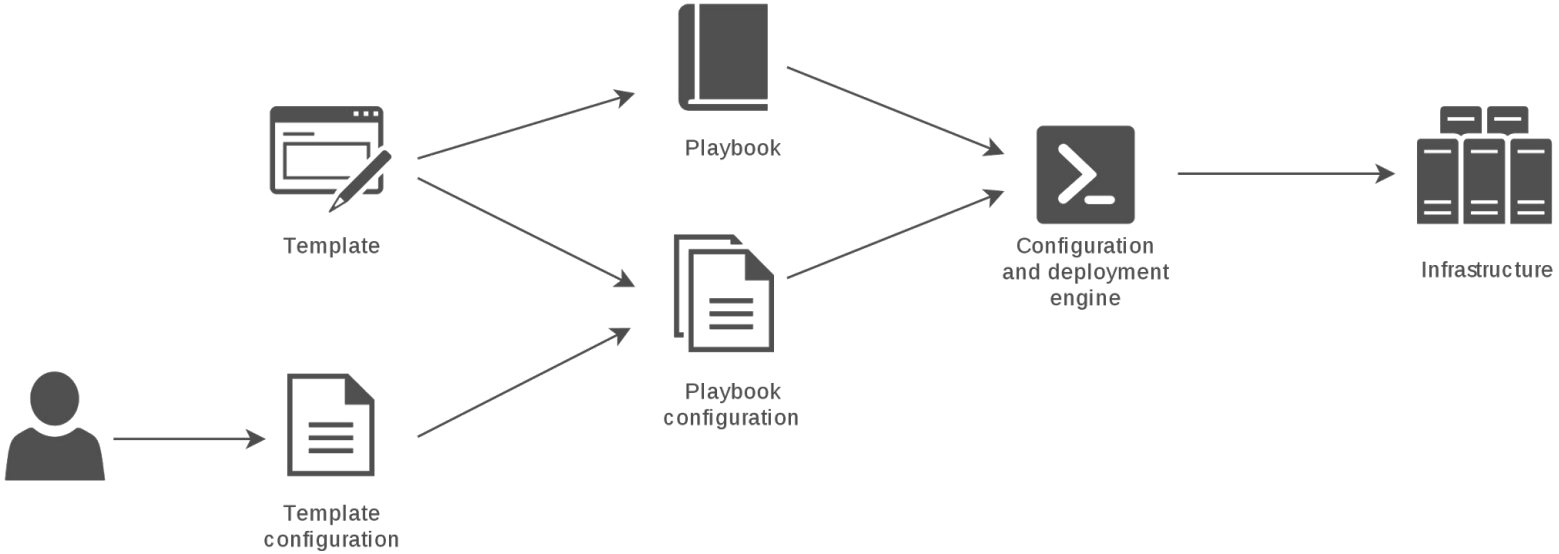
Business	SID
Customer (e.g. Escaux SA)	Project (escaux)
Product (e.g. MyPBX)	Template (my-pbx)

escaux

CONCEPTS
WORKFLOWS

escaux

workflow



escaux

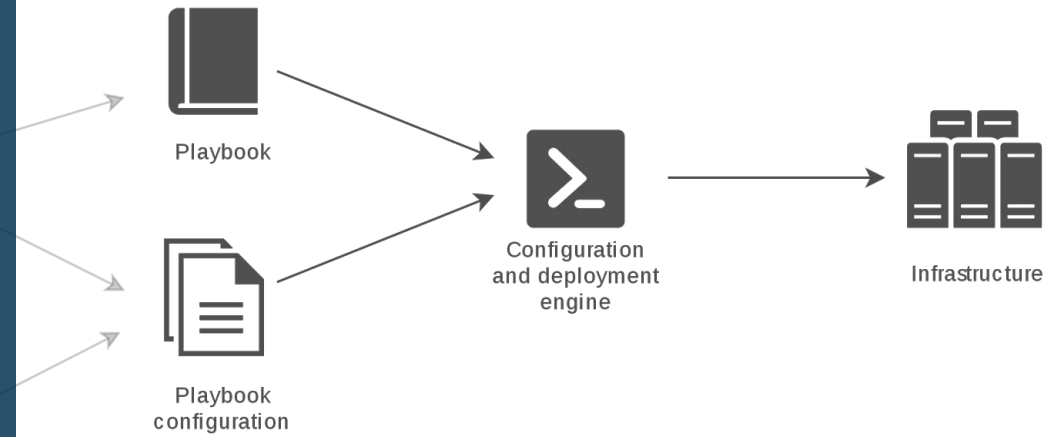
Workflow: Ansible

Playbook

Declarative process; a list of tasks (plays) to execute in given infrastructure context (group, host).

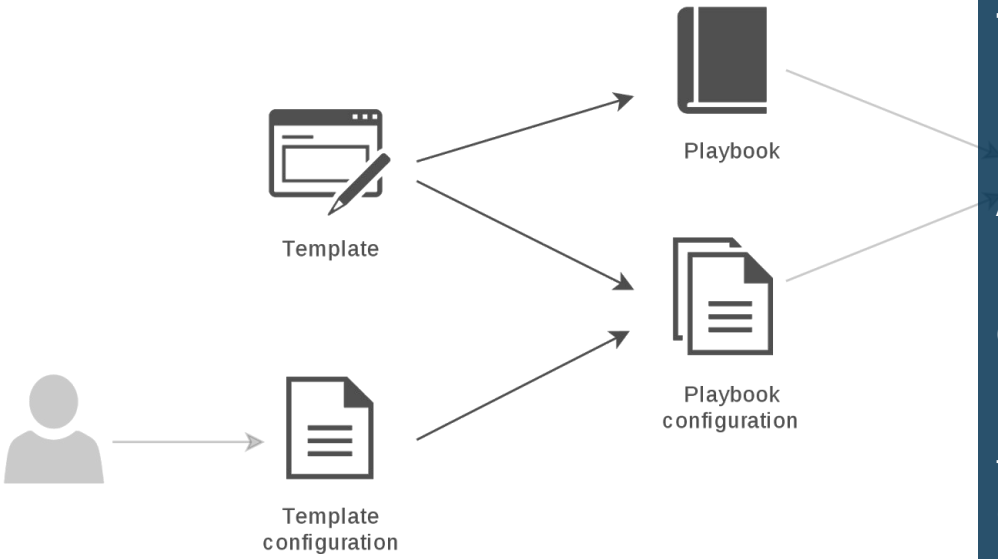
Configuration

Variables needed by the playbook to execute tasks: host_vars, group_vars.
E.g: Network configs



escaux

Workflow: Templating



Template

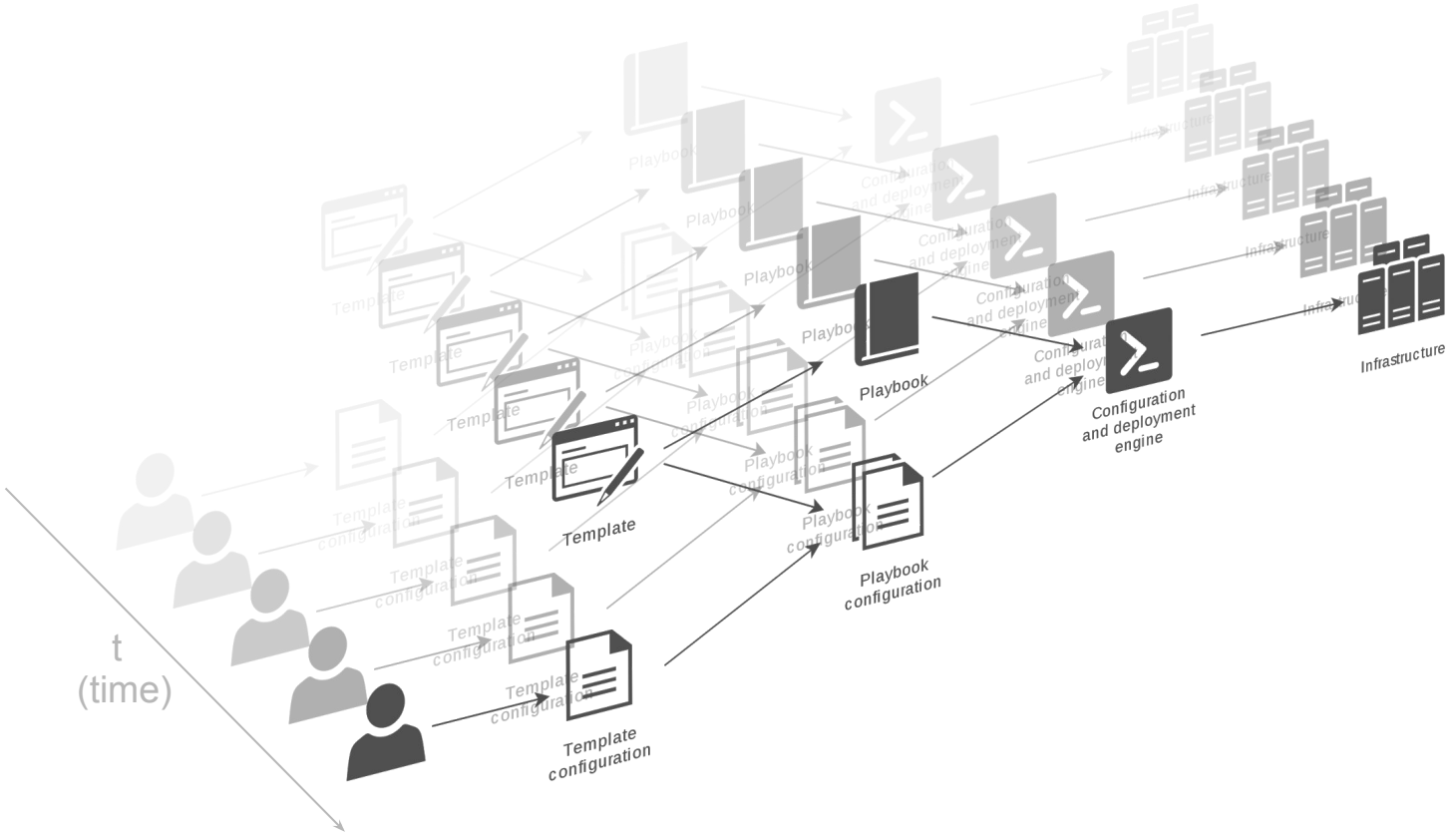
Directory tree; Jinja template of playbook and configuration files. Additional scripts and hooks.

Configuration

High-level constants needed by the template engine to generate files (e.g. customer ID).

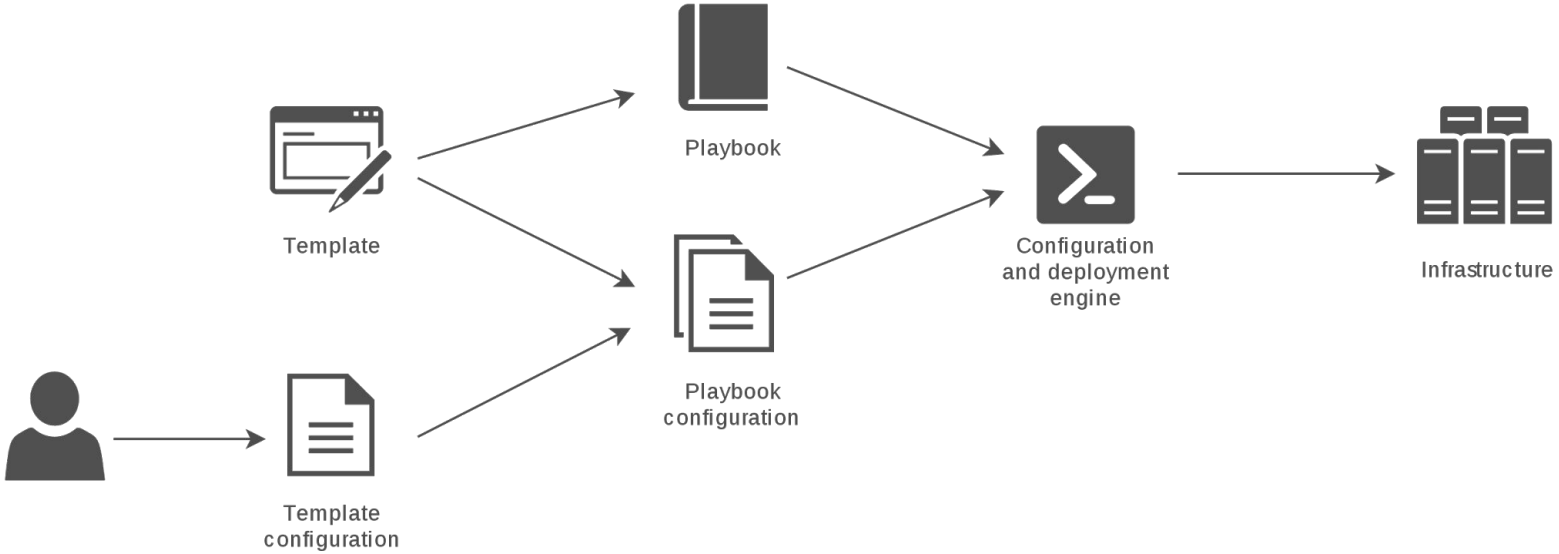
CONCEPTS
VERSIONING

versioning



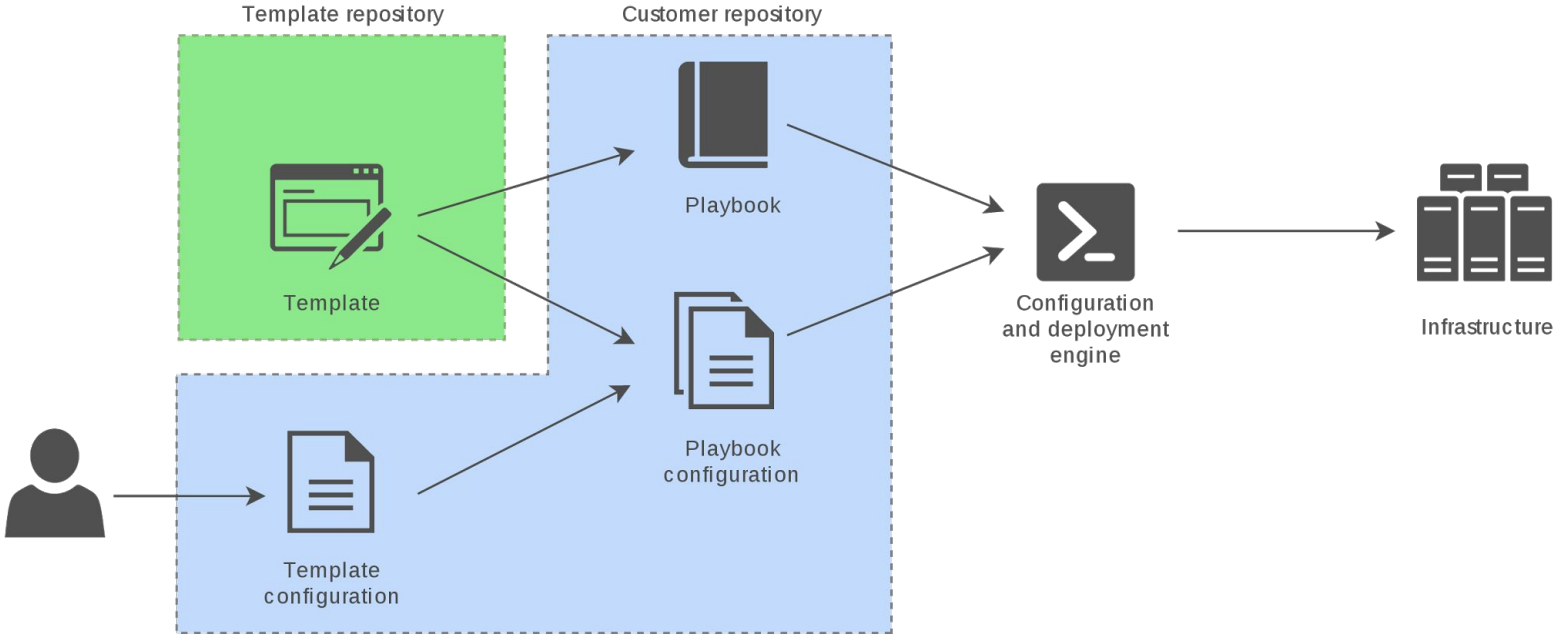
escaux

versioning



escaux

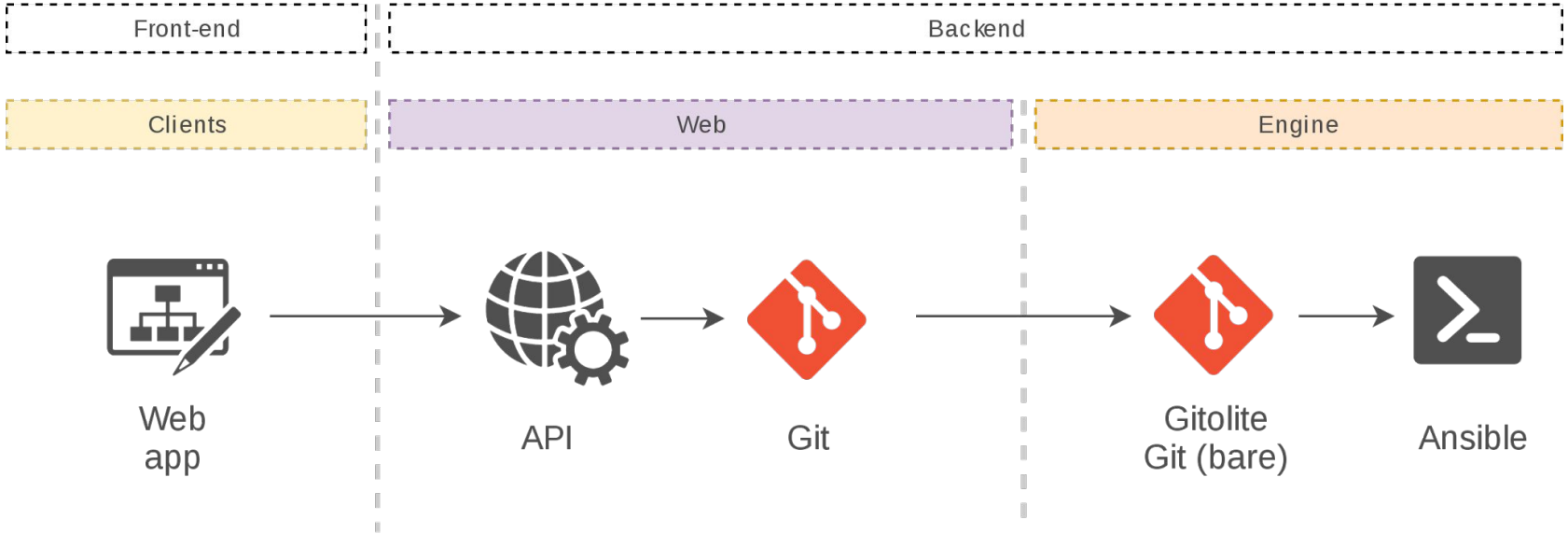
versioning



escaux

Architecture
High-Level

Architecture



escaux

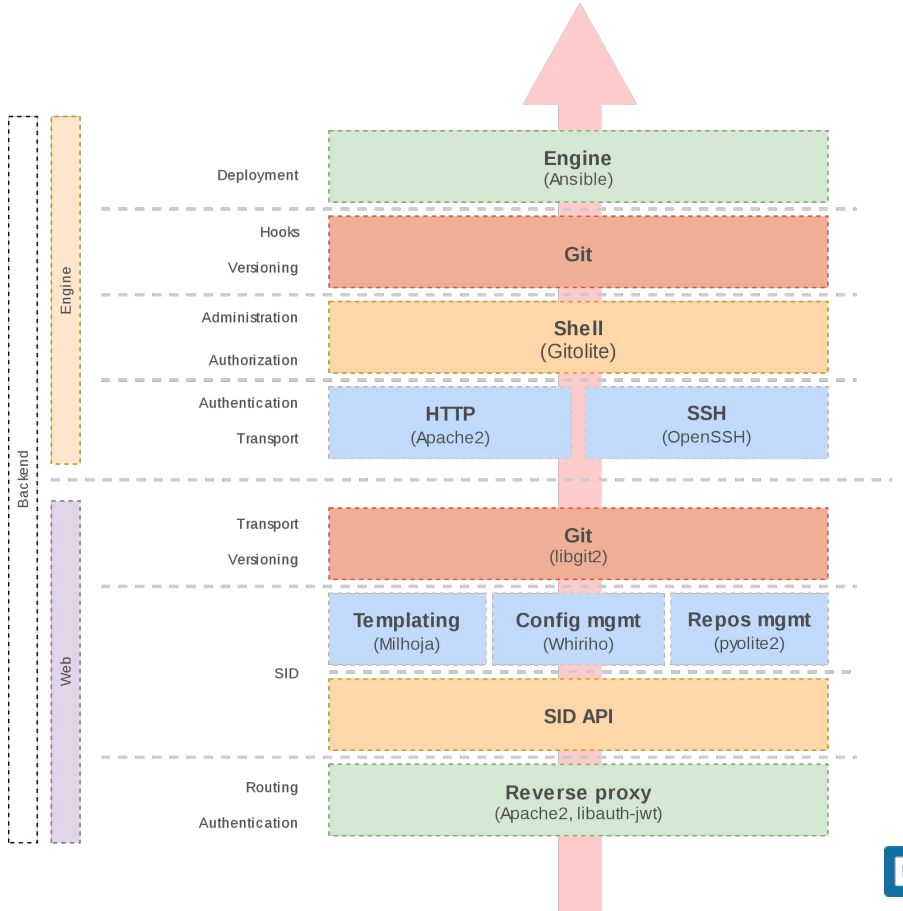
API

Features

- Versioning
- Content agnostic
- RESTful

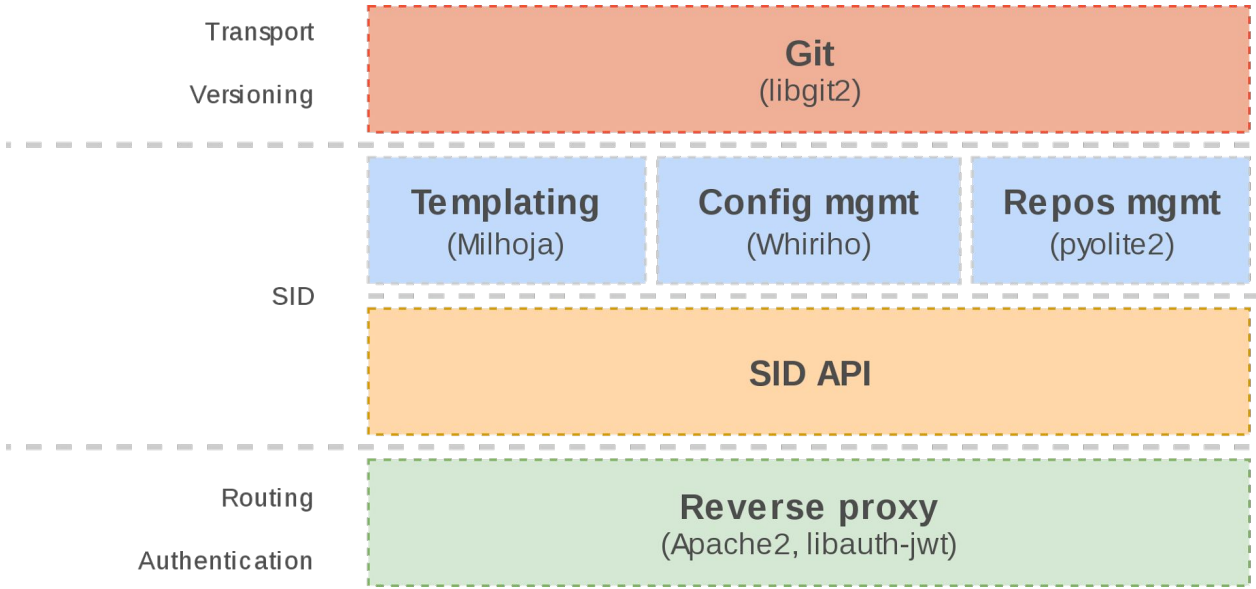
Thinking out loud

- SoC (Separation of concern)
- PFE (Proudly found elsewhere)

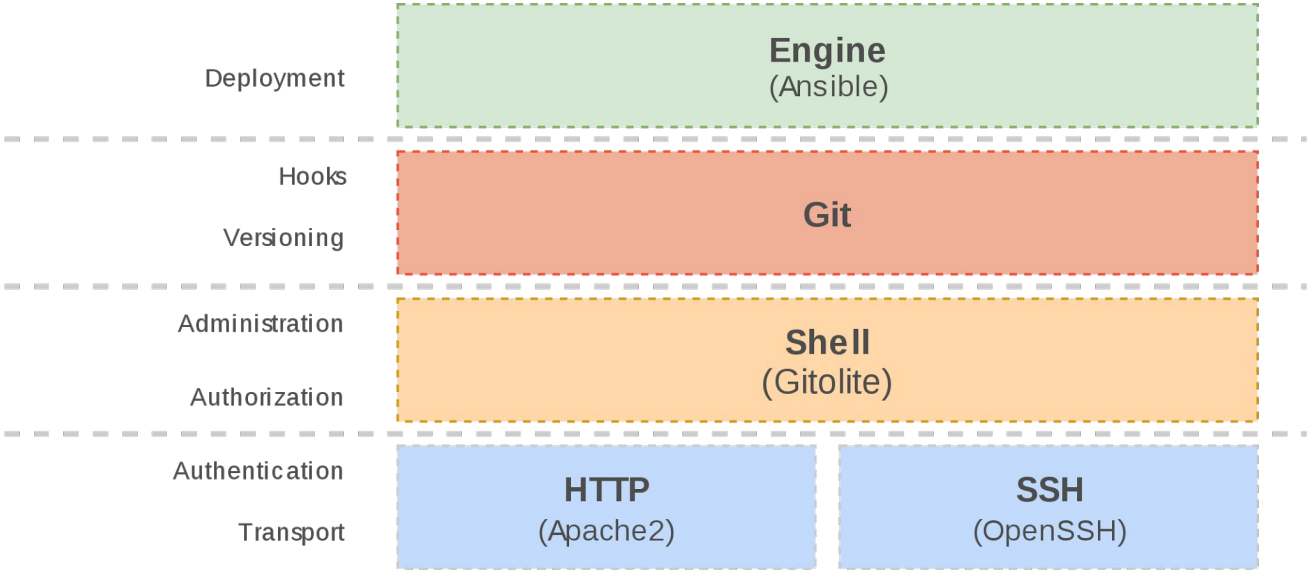


escaux

API: Frontend



API: backend


Deployment
Engine
(Ansible)
Hooks
Versioning
Git
Administration
Authorization
Shell
(Gitolite)
Authentication
Transport
HTTP
(Apache2)
SSH
(OpenSSH)

ARCHITECTURE
PROJECTS MANAGEMENT

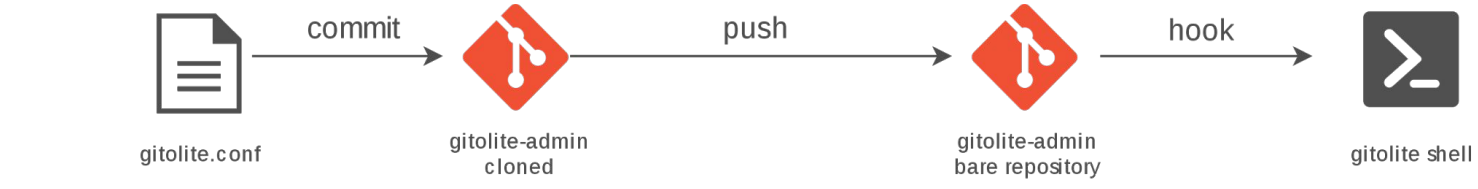
escaux

GITOLITE3

> Gitolite allows you to setup Git hosting on a central server, with very fine-grained access control and many (many!) more powerful features.

escaux

GITOLITE3




```
repo my-project
    RW john
    R  @all

repo another-project
    RW @all
```

escaux

create a new project

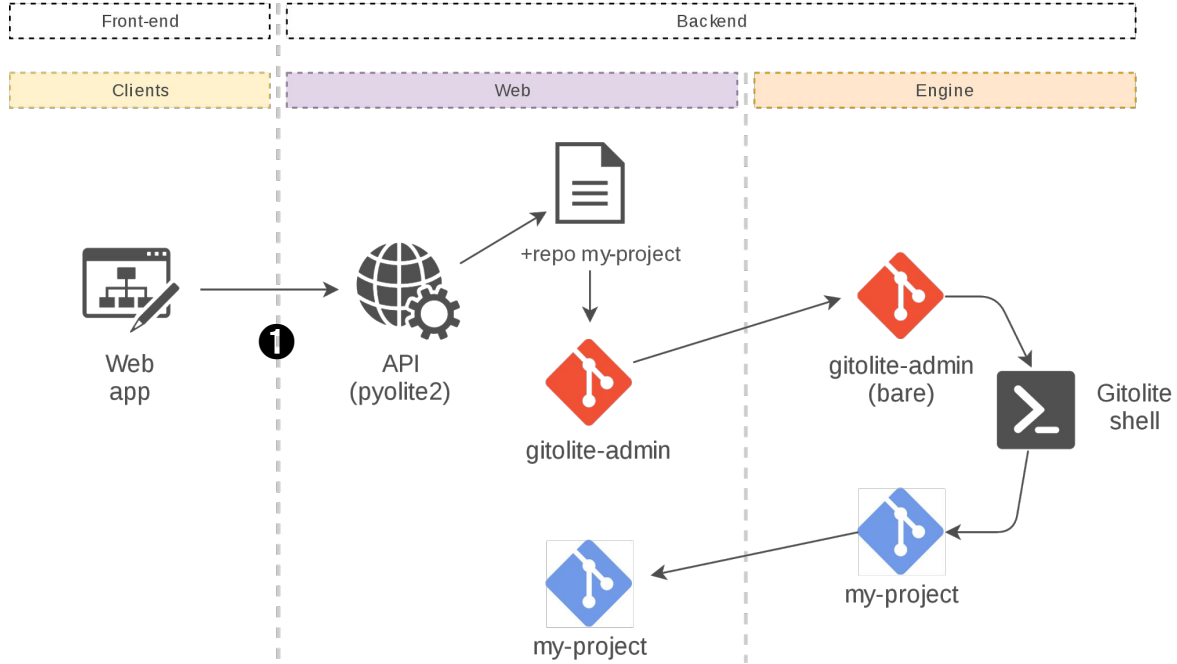
> I would like to use the API in order to create a new customer project.

escaux

pyolite2 in sid

❶ HTTP request

```
> POST /projects
> Host: sid.example.com
>
{
  "Name":"my-project"
}
```



escaux

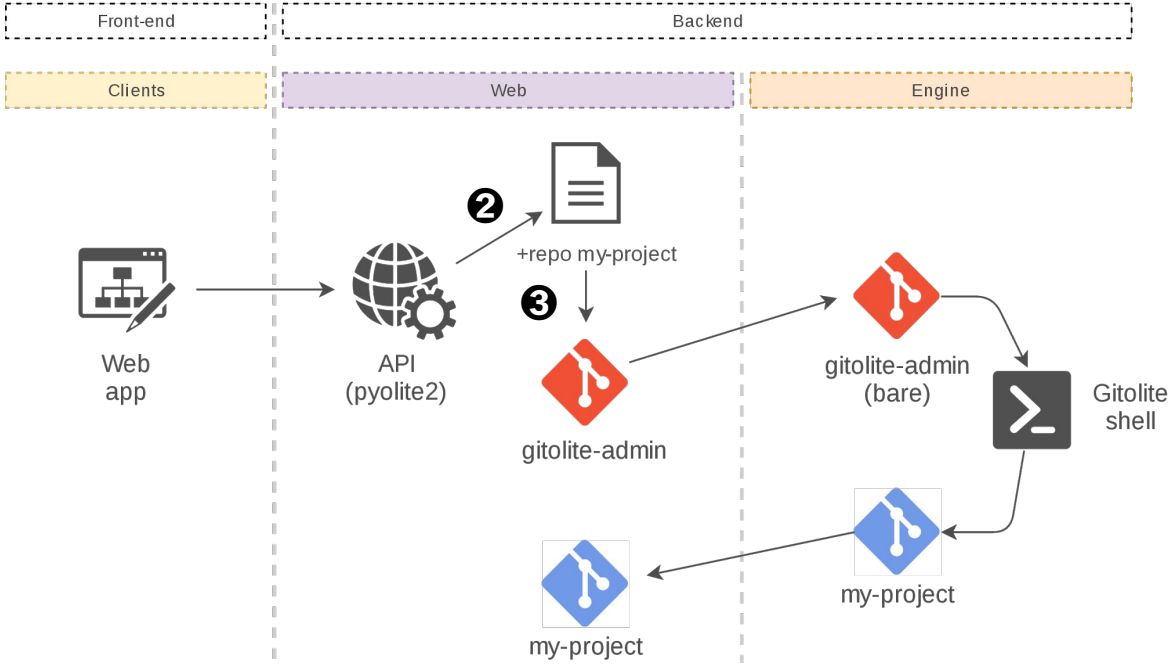
pyolite2 in sid

❷ Modify “gitolite.conf”

```
Pyolite2
  .load()
  .repos.append("my-project")
  .save()
```

❸ Commit “gitolite.conf”

```
gitolite-admin
  .add("gitolite.conf")
  .commit("Added repository")
```



escaux

pyolite2 in sid

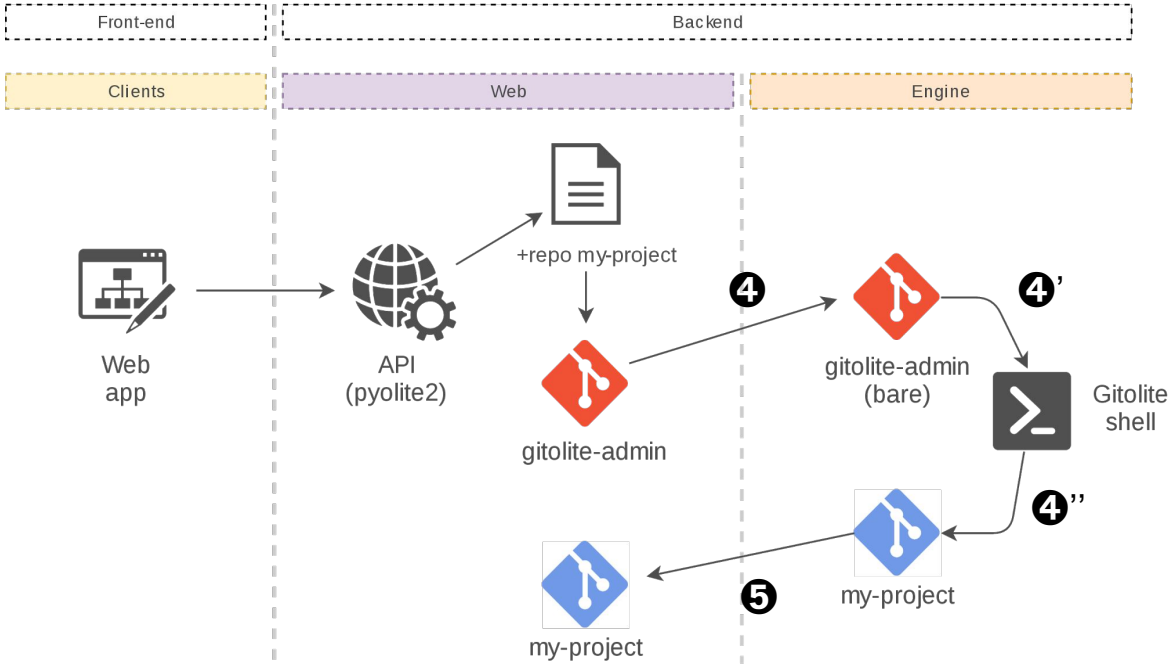
❹ Push gitolite-admin

By pushing gitolite-admin repository, it will trigger hook and recompile gitolite catalog.

Gitolite will detect missing repository “my-project” and will create it !

❺ Clone my-project

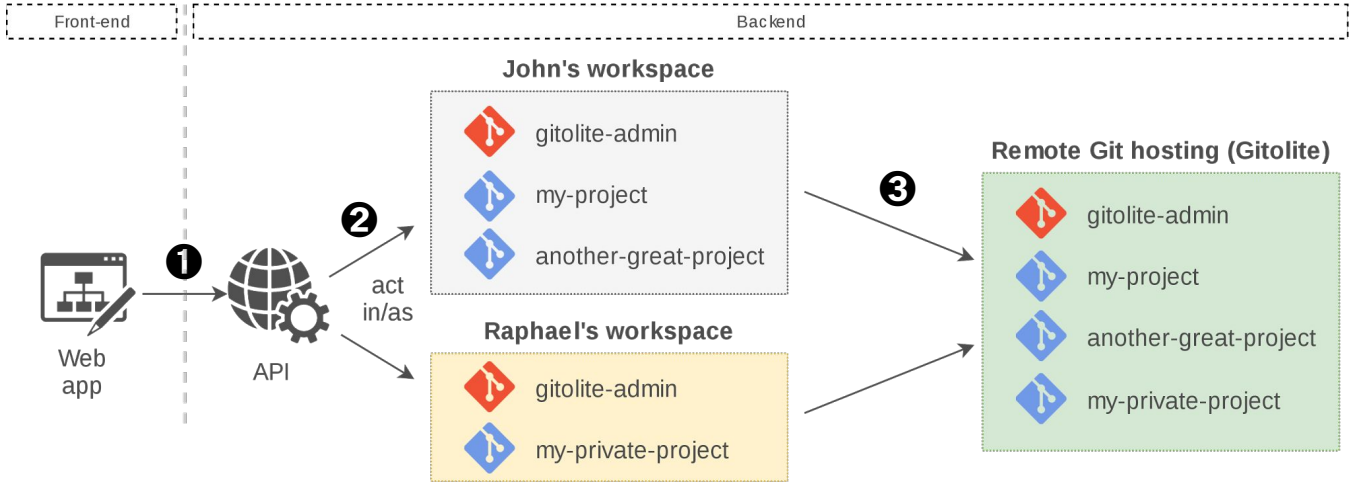
Now the repository exists on remote hosting, we can fetch it on our workspace.



escaux

Architecture workspaces

workspaces

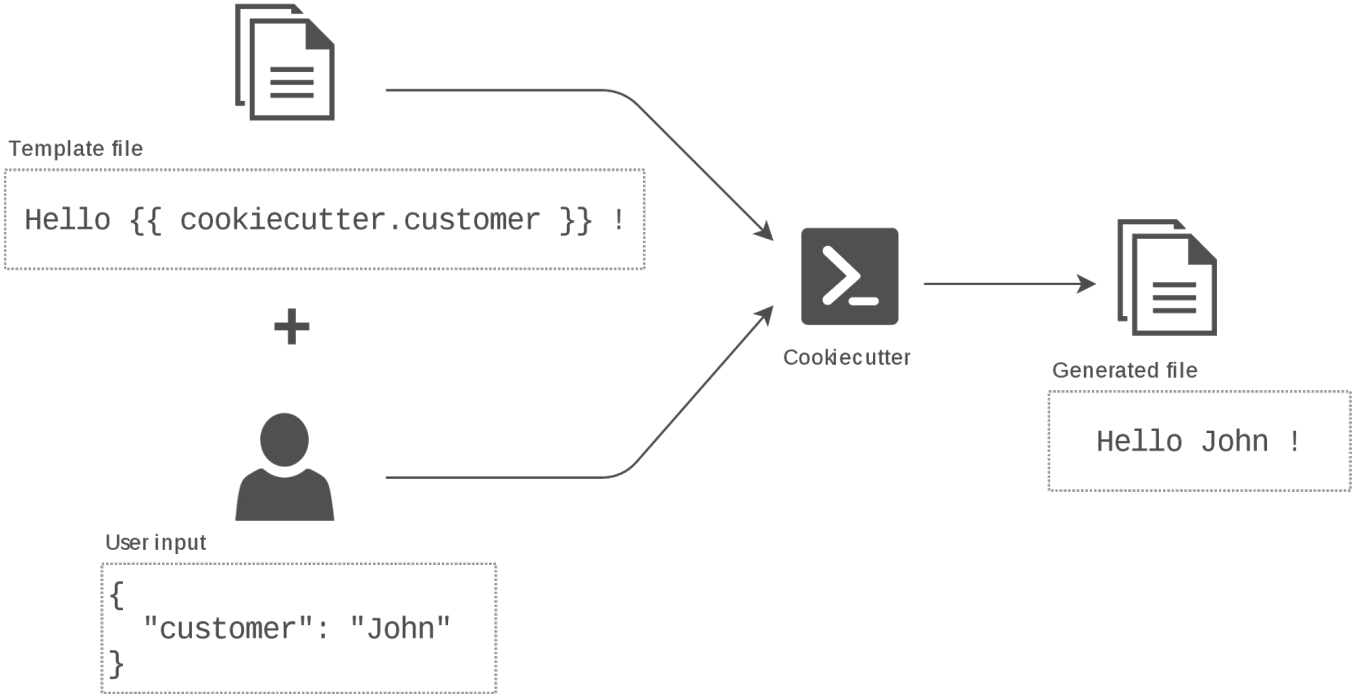


❶ HTTP requests authenticated through JWT
Extract user data from JWT
❷ Act as authenticated user in his own workspace
❸ Do remote Git operation using same JWT token (authentication forwarding)

escaux

Architecture
Templating

Templating: cookiecutter



escaux

Templating: cookiecutter

A demo is almost easier ...

Cookiecutter + Git = Milhoja

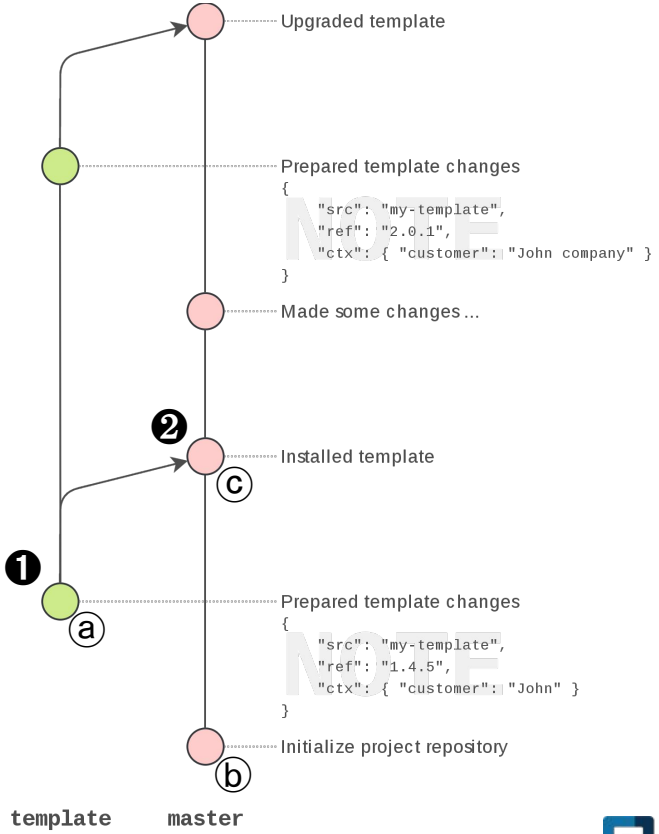
INSTALL a TEMPLATE

❶ “*Prepare template changes*”

- Create a temporary Git worktree
- Apply Cookiecutter template (user input) with user configuration (user input)
- Commit changes in template branch
- Create a note with template context
- Remove temporary worktree

❷ “*Install template*”

- In main worktree;
- Merge HEAD ⓑ with template branch ⓐ with priority to HEAD
- Commit changes ⓒ



escaux

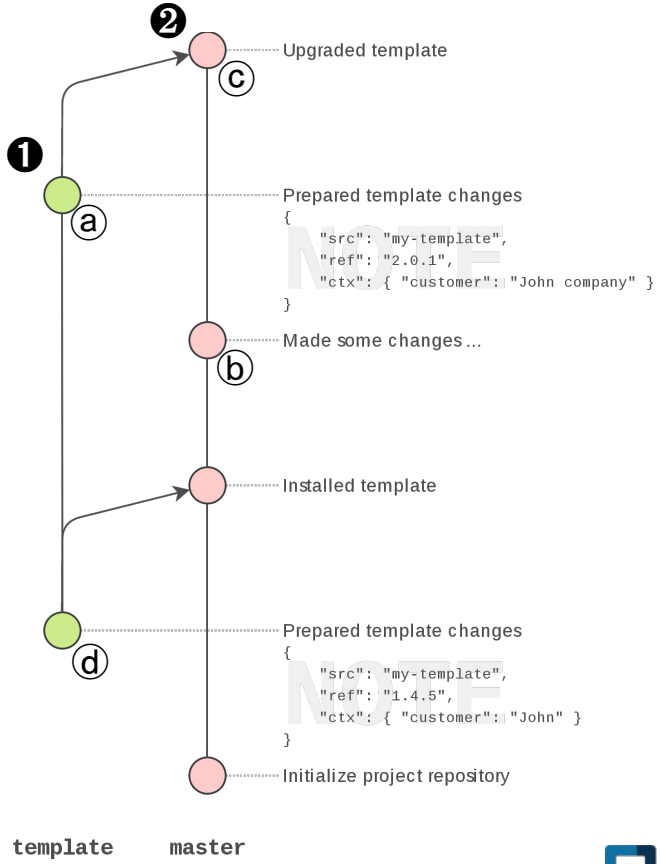
UPGRADE a TEMPLATE

❶ “*Prepare template changes*”

- Create an **EMPTY** temporary Git worktree
- Apply Cookiecutter template (context) with user configuration (context + user input)
- Commit changes in template branch with all staged files !!
- Create a note with template context
- Remove temporary worktree

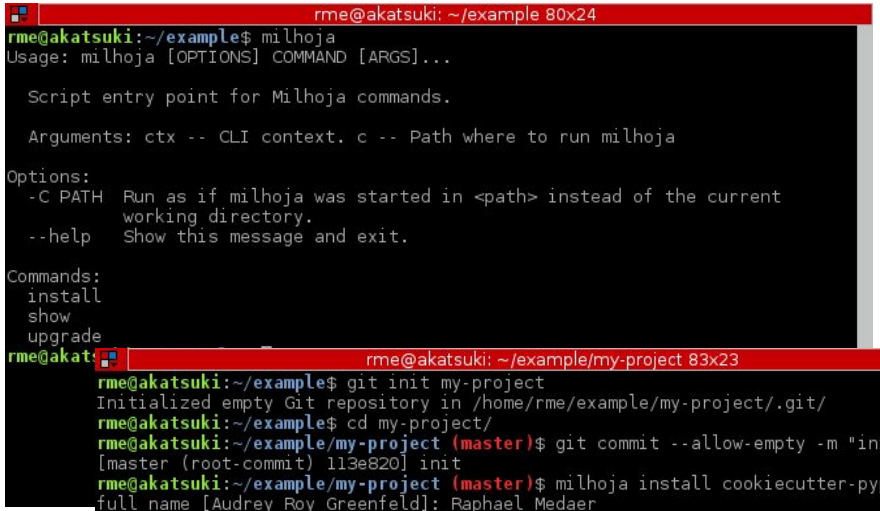
❷ “*Upgrade template*”

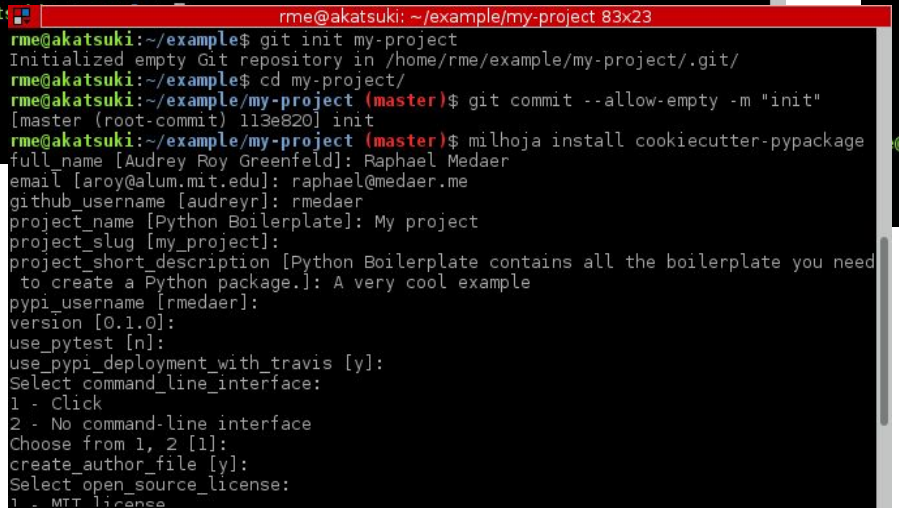
- In main worktree;
- 3-way merge between HEAD ⓑ, template branch ⓐ and common ancestor ⓓ
- Commit changes ⓒ

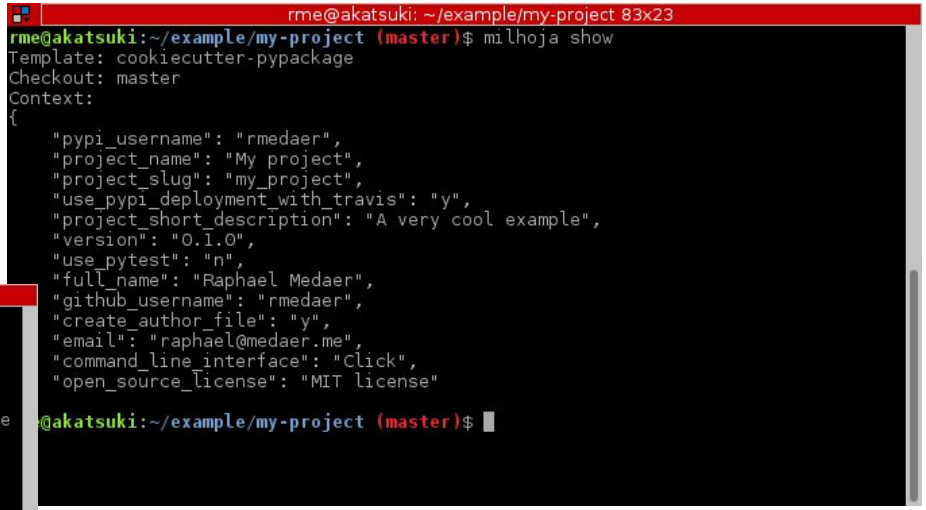


escaux

MILHOJA CLI







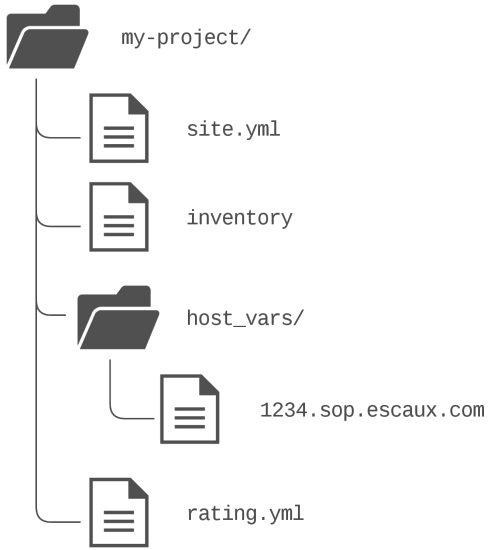
escaux

Architecture
Project configurations

Configuration files

Here is what your project looks like after templating.

Well ! Now how tell the API we're allowing user to edit "rating.yml" configuration and limit the edition to a strict configuration scheme ?



escaux

Whiriho

("Whirihoranga" means "Configuration" in Maori)

"A catalog of configuration files and their schemas"

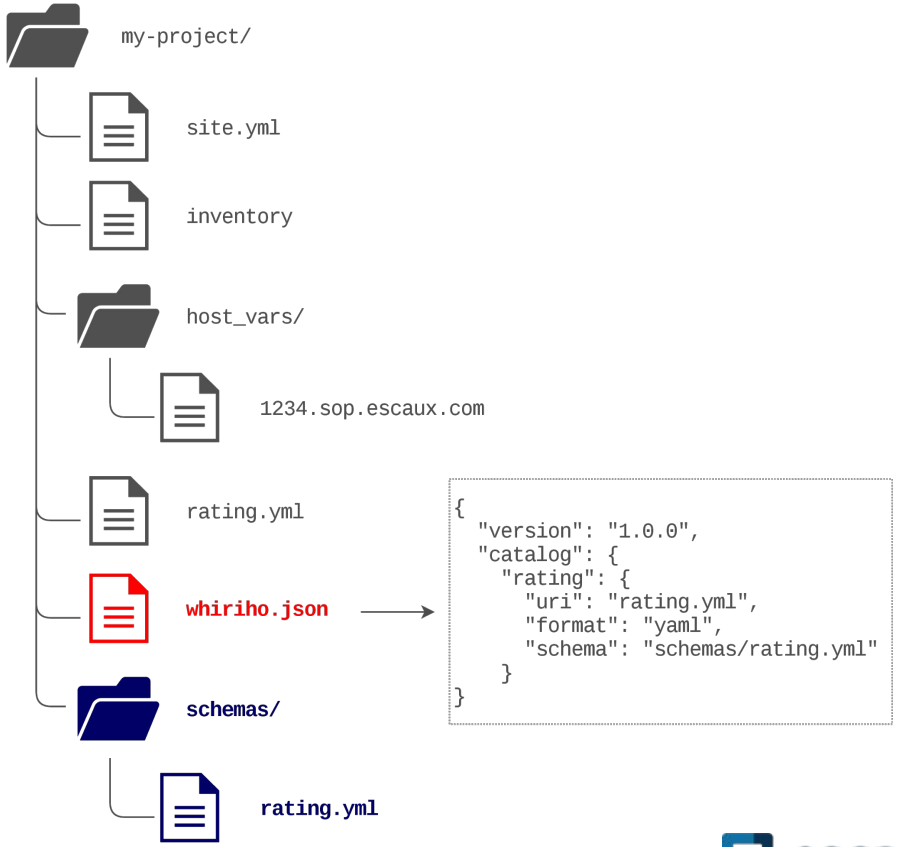
Configuration files

Whiriho **IS** ...

- a catalog of available configuration files
- linking each config file to a schema
- validating configuration with given schema
- a tool which abstract configuration format (yml, json, xml, …) using python-anyconfig from RedHat
- a Python library and a CLI

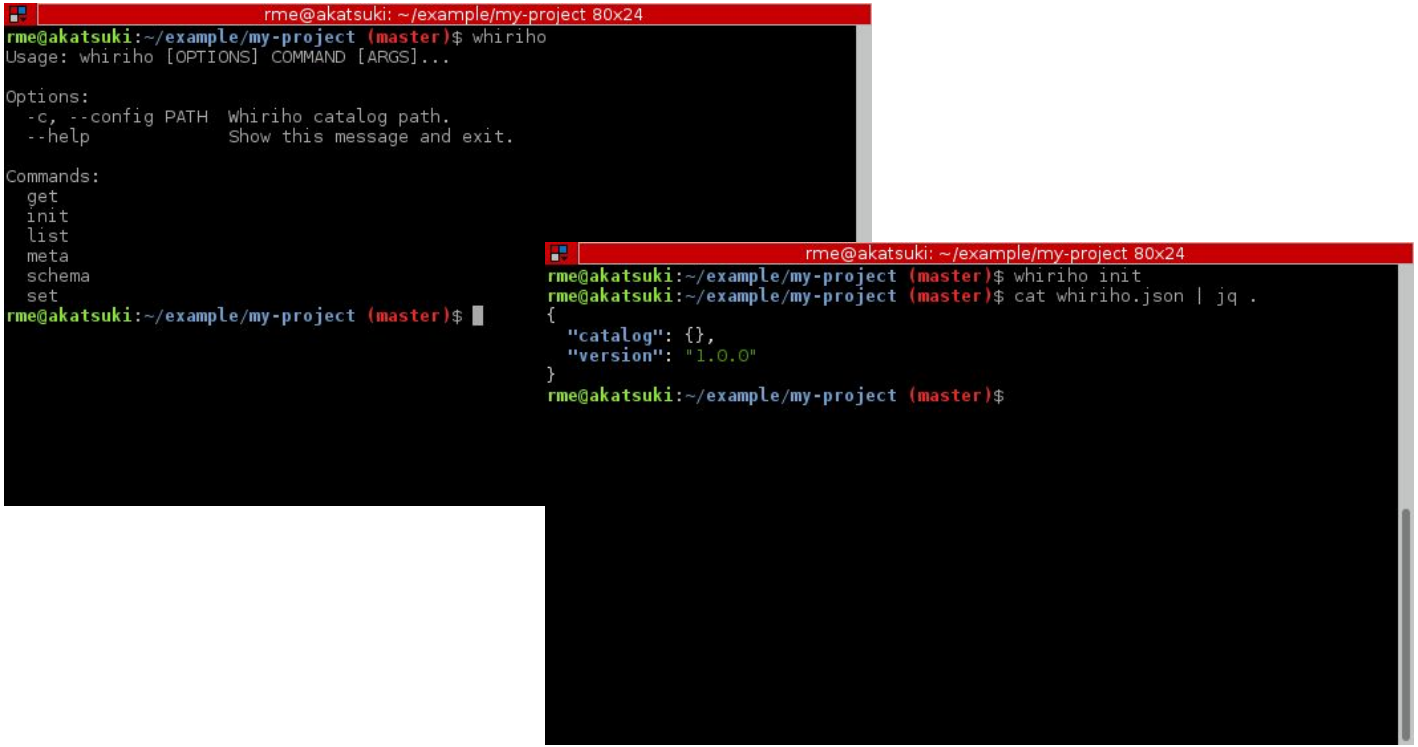
Whiriho **IS NOT** ...

- playing with Git
- managing permissions



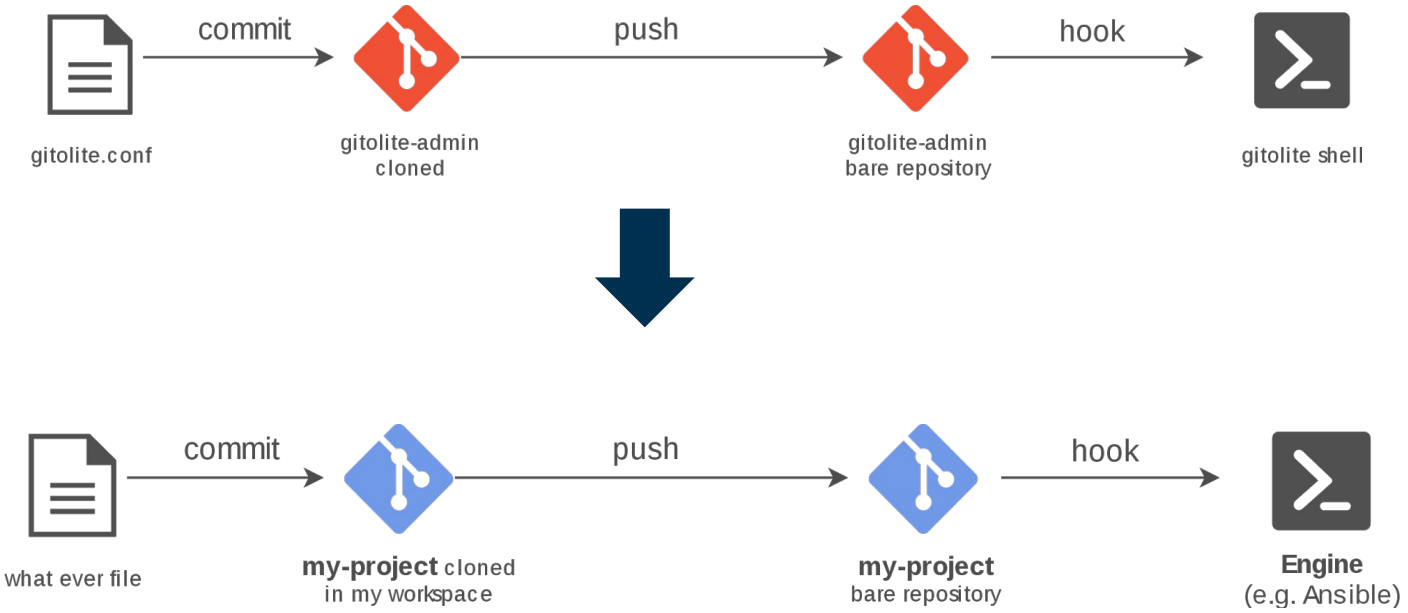
escaux

whiriho CLI


rme@akatsuki: ~/example/my-project 80x24
rme@akatsuki:~/example/my-project (master)$ whiriho
Usage: whiriho [OPTIONS] COMMAND [ARGS]...
Options:
-c, --config PATH Whiriho catalog path.
--help Show this message and exit.
Commands:
get
init
list
meta
schema
set
rme@akatsuki:~/example/my-project (master)$
rme@akatsuki: ~/example/my-project 80x24
rme@akatsuki:~/example/my-project (master)$ whiriho init
rme@akatsuki:~/example/my-project (master)$ cat whiriho.json | jq .
{
"catalog": {},
"version": "1.0.0"
}
rme@akatsuki:~/example/my-project (master)$

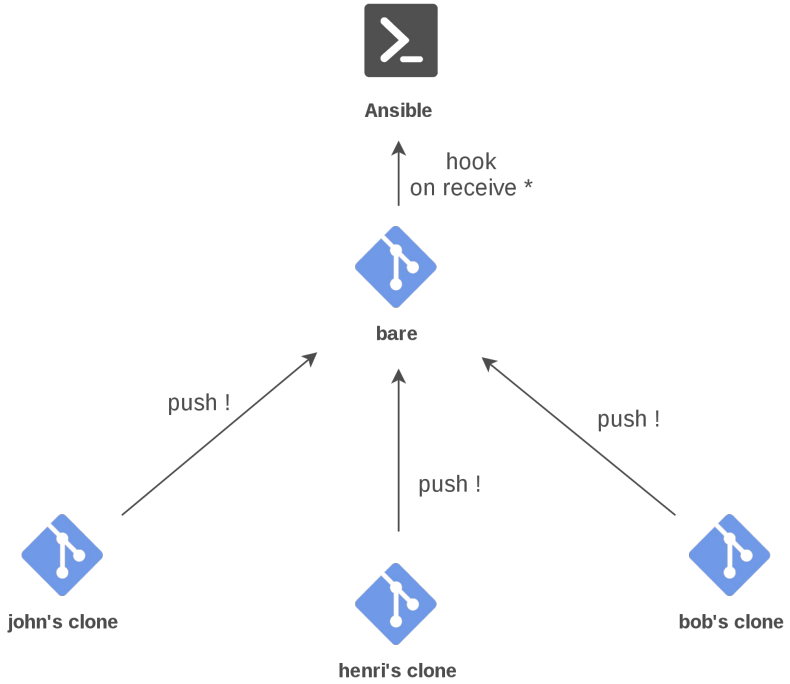
Architecture
Deployment

DEPLOYMENT: GITOLITE VS PROJECTS



escaux

DEPLOYMENT



“Tag each deployment”

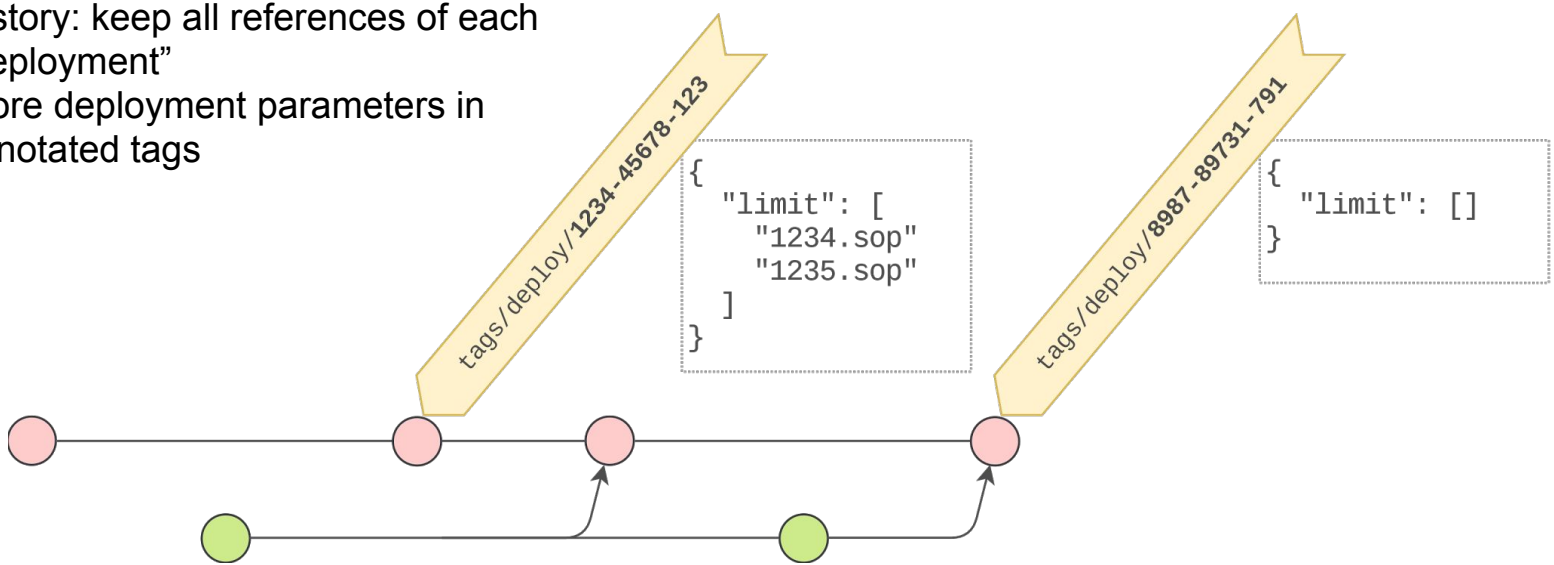
`refs/tags/deploy/<id>` → **deploy**

escaux

Deployment using tags

PRO tags:

- Point each deployment on targeted commit
- History: keep all references of each “deployment”
- Store deployment parameters in annotated tags



escaux

ABOUT THE API

API specifications

- Python (>= 2.7)
- HTTP framework (Tornado)
- Authentication (JWT)
- Internal libraries:
  - Whiriho (configuration files)
  - Milhoja (template)
  - Pyolite2 (gitolite)
- Libraries:
  - pygit2 (patched)
  - cookiecutter
  - jsonschema
  - jsonpatch

On GitHub

Fetch/clone and build using myrepos (mr) !

https://github.com/rmedaer/sid

```
git clone -b master git@github.com:rmedaer/sid
cd sid
echo "$PWD/.mrconfig" >> ~/.mrtrust

# Checkout libjwt debian packaging and build it
mr -d debian/libjwt checkout
mr -d debian/libjwt build

# Checkout all projects
mr checkout
```

WORKSHOP !

escaux

Workshop: API

1. Get Postman collection and environment example

   https://github.com/rmedaer/sid-api/tree/develop/docs
2. Fill environment with following data:
   - *Username*
   - *Password*
3. Generate a token (“Authentication” section)
4. Let’s get cracking !

thank you !

escaux