Once upon a time….

Automating Jenkins (re)installation:

some thoughts, tips, and tricks

Presentation available at: https://jmMeessen.github.io/slides/jw-eu-2019

Hello !!

- **Jean-Marc MEESSEN**
  - DevOps Consultant @ CloudBees
  - Belgian (beer and chocolates)

  - 🐦 @jm_meessen
  - ✉ jmeessen@cloudbees.com

Who are you?

What is configuration Management?

Pet versus Cattle

Automation, automation, automation!

In Source Control!

Why are CI/CD systems handled as Pet ?

Use cases

- **Provision** (bootstrap) new CI/CD cluster 🐣

- **Update** the system

    - ex: add a new master, change a setting, add a plugin

Other automation objectives

- Properly document the system

  - Peer-review and auditability

- Scaling

- support CI/CD power users

  - behind the scene warranty for creativity

Jenkins configuration vectors

Direct file manipulation

Command Line Interfaces

- two types

    - REST API

    - Jenkins CLI

REST API

- using HTTP requests to GET, PUT, POST and DELETE data.

```
curl -X POST "<jekinsURL>/testProject/build" --user jmm:<password|token>
```

REST API $\Rightarrow$ use a token

Jenkins CLI

Jenkins CLI - Classic

Much better ☻ :

```
java -jar jenkins-cli.jar -http -s $JENKINS_URL -auth @FILE command ...
```

Can use file permission, easy to configure with Ansible

Jenkins CLI - SSH

```
ssh -l jmm -i ~/.ssh/id_rsa -p 10200 my-jenkins-server help
```

🤔 Can solve some of the bootstrapping problem

Summary

- Rich set of API

- Easy to use in Ansible for example

- CLI does a better job at controlling parameter

- CLI makes synchonous calls

- CLI commands are better documented 😇

- Parsing results is tricky 😖

Recommendation

- Use CLI

- Use CLI with SSH if you can (networking)

- Consider executing commands from target host (localhost).

Groovy Scripts
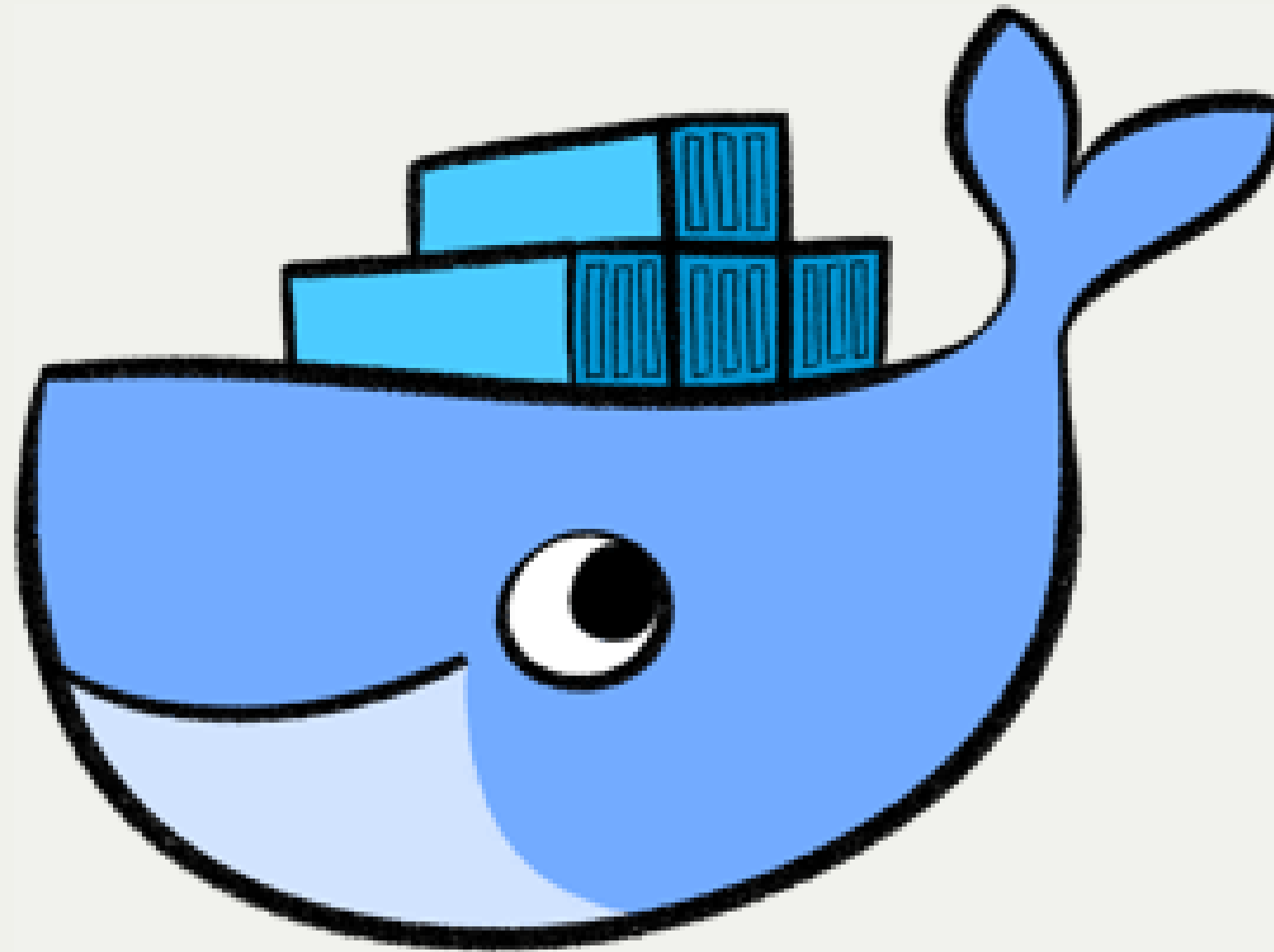

- Make them idempotent ! 👀

How to use Groovy Script

- via the script console

- at startup, as init-script

    - placed in `$JENKINS_HOME/init.groovy.d/`

- via the CLI

Groovy Scripts from the CLI

```
cat my_script.groovy | {{ CLI_command }} groovy =
```

Docker Container

Jenkins Configuration as Code

- Declarative method, yaml based

- Loaded on reboot or with a CLI command

JCasC Example (LDAP cfg)

```yaml
jenkins:
  securityRealm:
    ldap:
      configurations:
      - inhibitInferRootDN: false
        managerDN: "uid=idm,ou=Administrators,dc=example,dc=com"
        managerPasswordSecret: "{{ ldap_admin_passw }}"
        rootDN: "dc=example,dc=com"
        server: "ldap://{{ full_agent_docker_dns_name }}:389"
      disableMailAddressResolver: false
      disableRolePrefixing: true
      groupIdStrategy: "caseInsensitive"
      userIdStrategy: "caseInsensitive"
```

JCasC Example (JNLP agent)

```
jenkins:
  nodes:
  - permanent:
      labelString: "jnlp"
      mode: NORMAL
      name: "jnlp-agent"
      remoteFS: "/home/jenkins"
      launcher:
        jnlp:
          workDirSettings:
            disabled: true
      nodeDescription: "Agent that initiates its own connection to Jenkins"
      retentionStrategy: "always"
  numExecutors: 0
```

Current Status

- In technical preview for CloudBees products
    - Waiting for RBAC support 😛
- Centralized CasC management from CJOC

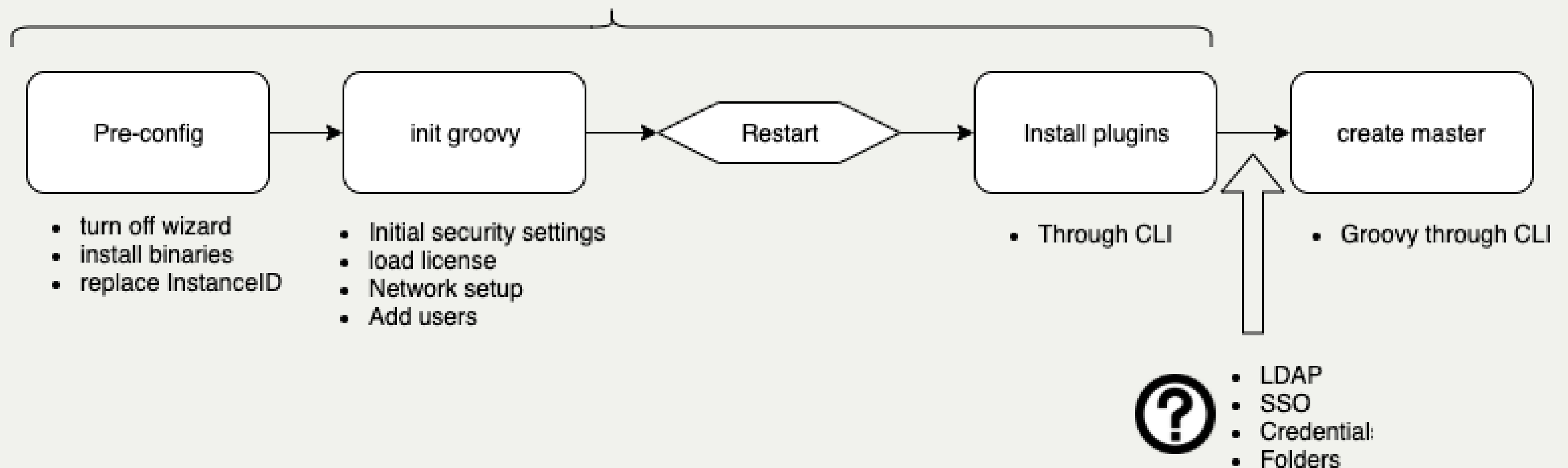# And in Real Life?
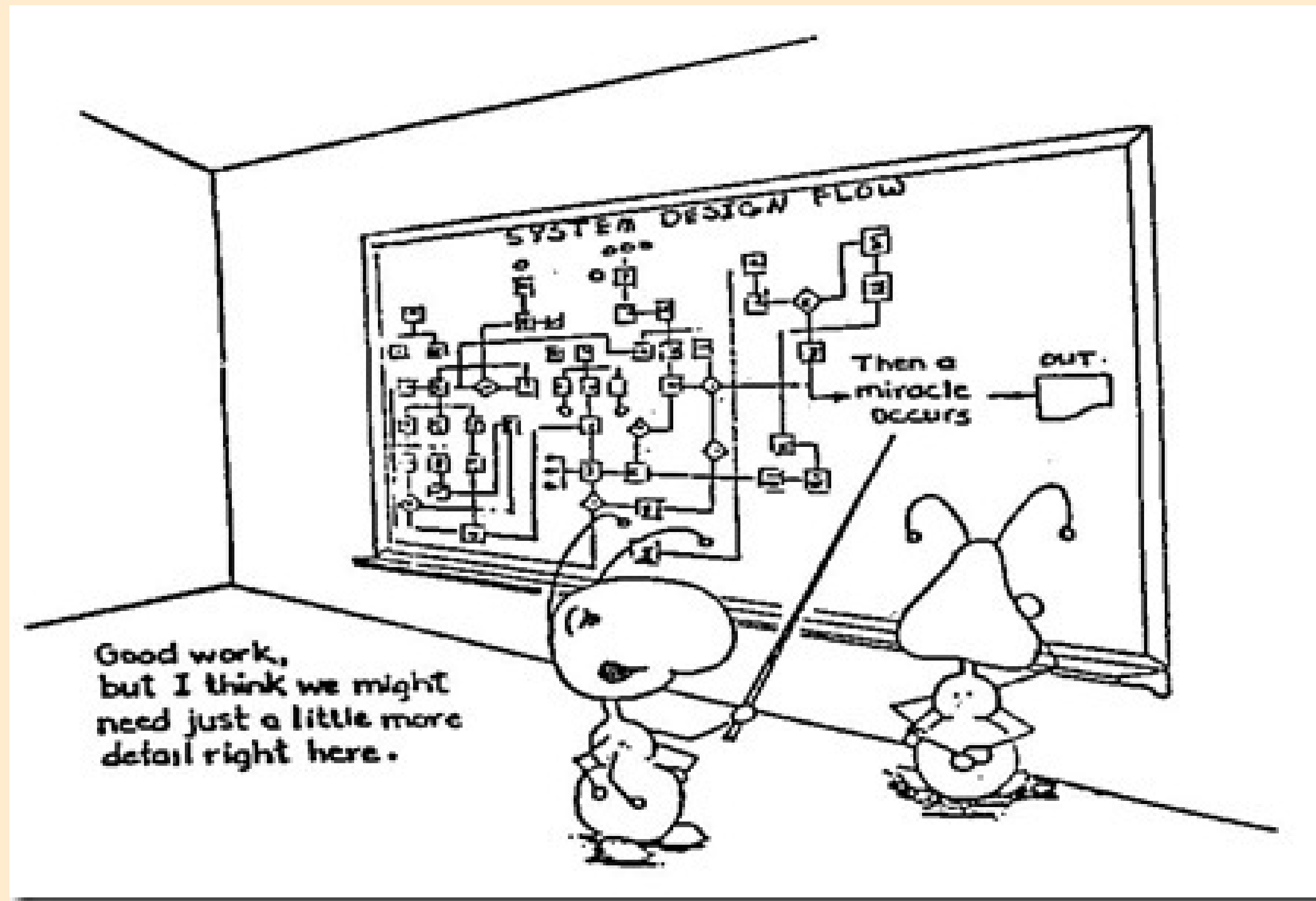
My opinion

- Practice is still "sedimentary config layer"

- No easy way to solve bootstrapping problem

- Poorly documented / tooled

- Not fit for the volatile K8S world
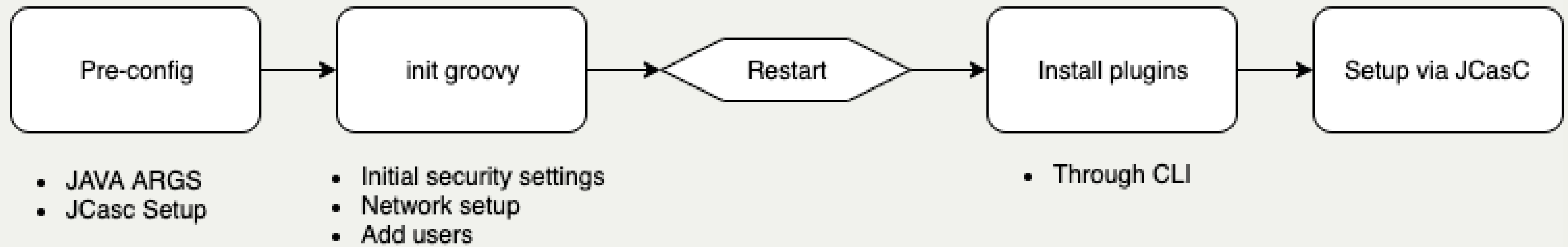
But Cloudbees is actively working on it

Bootstrap strategy

- see example on **https://github.com/jmMeessen/captains_aws_cjp**

CJOC Bootstrap

Pre-config → init groovy → Restart → Install plugins → create master

Pre-config:
- turn off wizard
- install binaries
- replace InstanceID

init groovy:
- Initial security settings
- load license
- Network setup
- Add users

Install plugins:
- Through CLI

create master:
- Groovy through CLI

- LDAP
- SSO
- Credentials
- Folders

Add Master

```
┌─────────────┐     ┌─────────────┐    ╱──────────╲     ┌─────────────┐    ┌─────────────┐
│  Pre-config │ ──▶ │  init groovy│──▶ ╲  Restart  ╱──▶ │Install plugins│──▶│Setup via JCasC│
└─────────────┘     └─────────────┘    ╲──────────╱     └─────────────┘    └─────────────┘
```

- JAVA ARGS
- JCasc Setup

- Initial security settings
- Network setup
- Add users

- Through CLI

Good to know

Plugins loaded → JCasC → Jobs loaded → Init Groovy → Done

Some thoughts

Thoughts (1)

- AUTOMATE! (especially in a Cloud World)

- "CasC" is the way to go …

- Plugins installation !

- Bootstrapping is not solved yet

  - But not a reason not to start now

Thoughts (2)

- Exercise your system (drift…)

- all config changes must be done via source & automation

- Cut the access to that administration UI ☜

And a last one 😉

- Self-Service portals…
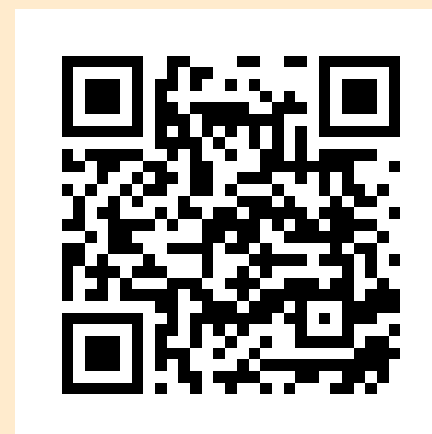   - very 2010
    - why not ask for a configuration PR?

Thank You !

🐦 @jm_meessen

⑂ jmMeessen

Slides: https://jmMeessen.github.io/slides/jw-eu-2019

Source on ⑂: https://github.com/jmMeessen/slides/tree/jw-eu-2019

Bonus

REST API documentation

- To learn more: 📖

  - https://wiki.jenkins.io/display/JENKINS/Remote+access+API

  - https://wiki.jenkins.io/display/JENKINS/Authenticating+scripted+clients

BONUS - how to retrieve a token

```
# First thing we do is obtain a crumb from cjoc, this allows us to call the rest API wi
JENKINS_CRUMB=$( \
  curl --user $USERNAME:$PASSWORD \
      --silent "$URL/cjoc/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,\":\",/
)

# Get the JSON payload from the generateNewToken API
JENKINS_TOKENS_JSON=$( \
  curl --header "$JENKINS_CRUMB" \
      --user $USERNAME:$PASSWORD \
      --silent "$URL/cjoc/user/$USERNAME/descriptorByName/jenkins.security.ApiTokenPro
      --data "newTokenName=$USERNAME" \
)

# Pulls the token out of the JSON payload.
```

Jenkins CLI - More details

- https://jenkins.io/doc/book/managing/cli/

Groovy Scripts

- See this *Knowledge Base article* on how to access the javadocs

JCasC

- https://github.com/jenkinsci/configuration-as-code-plugin

Bootstrap strategy

- see example on **https://github.com/jmMeessen/captains_aws_cjp**

1. Install jenkins configuration file (startup option)

    1. JAVA_ARGS → `-Djenkins.install.runSetupWizard=false`

2. Proceed with installation via package manager (`apt-get`)

3. Create `init.groovy.d` directory

4. Replace the instanceID with know one (`secret.key`)

Bootstrap strategy (cont.)

1. Add "init groovy scripts" in directory

    1. Initial security settings

    2. License loading script

    3. Set-URL, JNLP, and SSHD Port configuration scripts

    4. Create Cfg-Management user, generate key and load public key

2. Restart CJOC to activate scripts

3. Use CLI to install plugins

4. Use CLI to execute groovy to create Client Master

Bootstrap strategy (cont.)

1. Configure Client Master in same principle

    1. Add to JAVA_ARGS the connection info

    2. Configure security and initial users via init scripts

    3. Install default plugins

2. Configure JCasC environment

3. Copy definition in adequate directory

4. Use CLI to force the load of configuration