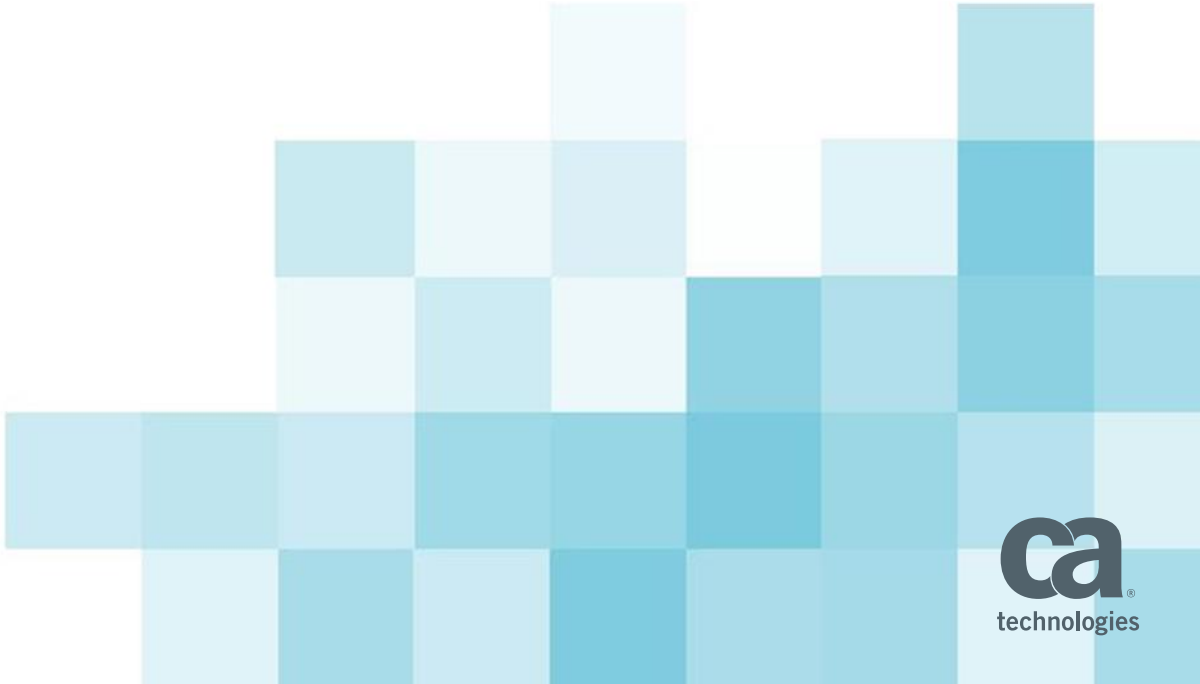
CA Release Automation Community Webcast Series

Creating a Release Automation Continuous Delivery Edition Plug-in

Walter Guerrero

June 22, 2016



ca technologies

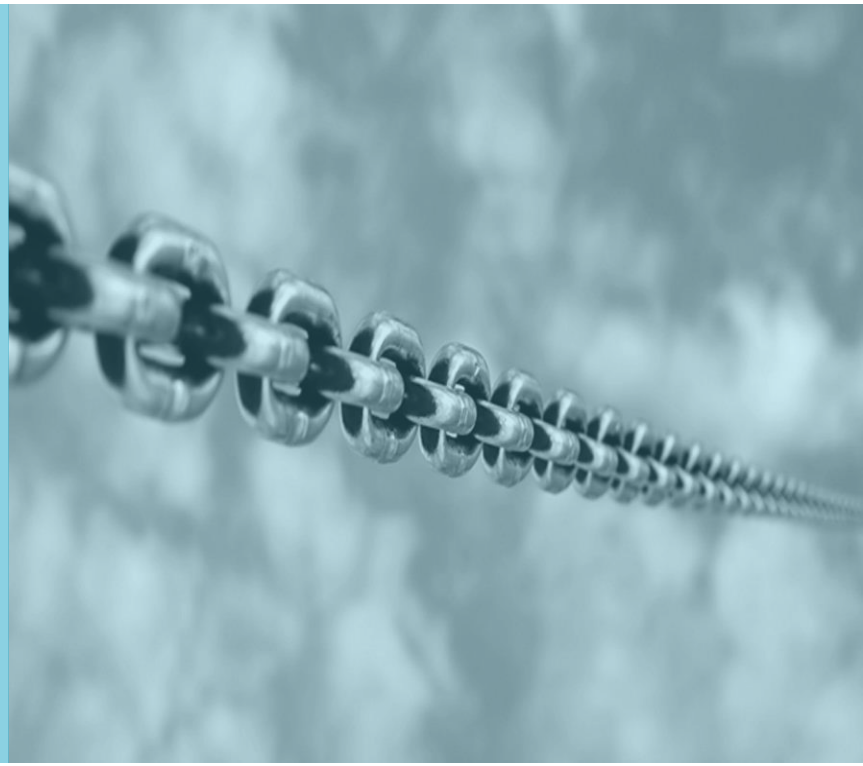
What We'll Cover in Today's Webcast

- RA CDE Plugin Framework
- Online micro services
- Offline services
- Manifest File
- Implementation
- Best Practices

Challenges of Building a Continuous Delivery Tool Chain

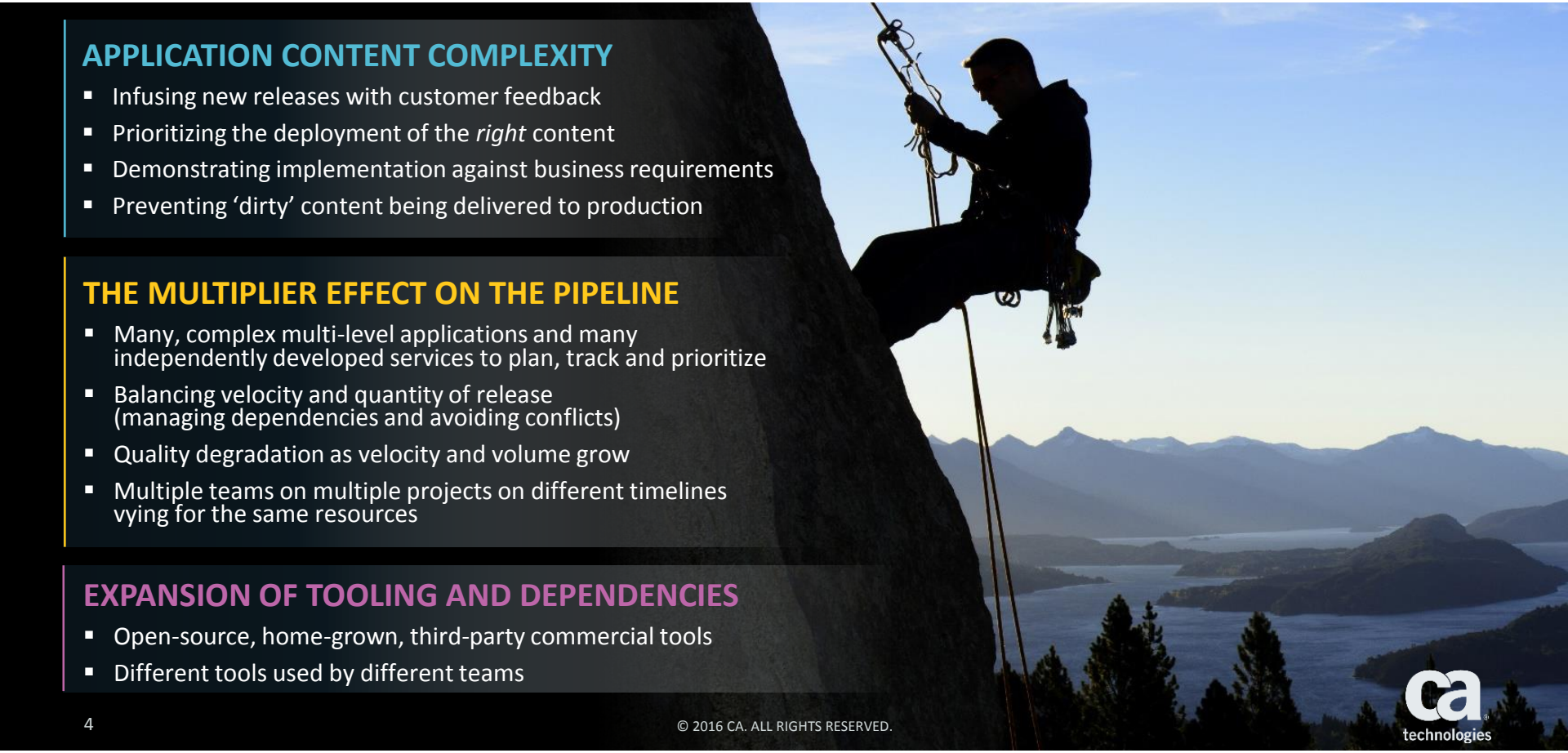
APPLICATION DELIVERY IS COMPLEX

- 100's, 1000's, 10,000's machines
- Provision and Configure
- Middleware, Databases, Load Balancers
- Applications with 100 of artifacts in multiple repositories
- Short release cycles
- Multiple releases per day/week/month
- Balancing speed with quality



ca technologies

New Pressure Points on Continuous Delivery Pipeline



APPLICATION CONTENT COMPLEXITY

- Infusing new releases with customer feedback
- Prioritizing the deployment of the *right* content
- Demonstrating implementation against business requirements
- Preventing 'dirty' content being delivered to production

THE MULTIPLIER EFFECT ON THE PIPELINE

- Many, complex multi-level applications and many independently developed services to plan, track and prioritize
- Balancing velocity and quantity of release (managing dependencies and avoiding conflicts)
- Quality degradation as velocity and volume grow
- Multiple teams on multiple projects on different timelines vying for the same resources

EXPANSION OF TOOLING AND DEPENDENCIES

- Open-source, home-grown, third-party commercial tools
- Different tools used by different teams

4

What is the CDE Plug-in Framework?

- The effectiveness of CA Release Automation Continuous Delivery Edition depends on its ability to interact with the remote components in your continuous delivery pipeline
- Supports
  - Configuration of Endpoints
  - Creation of automated tasks
  - Importing content

Plug-in Framework Architecture

- The architecture of the plug-in framework is intended to allow for quick and flexible development of integrations with remote components in your continuous delivery pipeline
- The plug-in framework does not require any specific programming language or delivery model
  - As an offline Java project that you install (similar to the current packaged plug-ins)
  - As an online service that the product connects to

Developing Plug-in

- To develop a custom plug-in, you must adhere to the following requirements:
  - The plug-in must be an HTTP service that can accept a POST request, instrument the requested operation, and return a response.
  - The plug-in must include a manifest.json file that details the plug-in capabilities.
  - After you finish plug-in development, you will follow the standard procedure to add the plug-in to RA CDE

Plug-in Manifest.json

- Describes the plug-in capabilities
- Makes the capabilities available as configurable capabilities
- The format to be followed
  - Basic plug-in information, such as name and version
  - Endpoint template that describes the name and required parameters
  - Information for each service that the plug-in provides, such as tasks, content import, or application model import
- The manifest file needs to be placed at the top level of the project

Typical Manifest.json

```
{
    "name": "Bamboo Plugin 1.0",
    "vendor": "CA technologies",
    "uniqueId": "ca.cde.bamboo",
    "description": "CA Release Automation Bamboo Plugin powered by Hook.io",
    "iconUrl": "https://cloud.githubusercontent.com/assets/14964166/12397368/7823d66e-be15-11e5-9b94-86673ff64912.png",
    "version": "1.0",
    "relativeUrl": true,
    "endpointTemplate": {
    "name": "Bamboo - Build",
    "uniqueId": "ca.cde.bamboo.build.endpoint",
    "description": "Bamboo Endpoint",
    "serviceType": "ENDPOINT",
    "endPointType": "Bamboo",
    "parameters": [{
    "name": "user",
    "uniqueId": "ca.cde.bamboo.endpoint.user",
    "displayName": "Bamboo User",
    "type": "string",
    "isOptional": false
    },
    {
    "name": "Password",
    "uniqueId": "ca.cde.bamboo.endpoint.password",
    "displayName": "Bamboo Password",
    "type": "password",
    "isOptional": false
    },
```

Typical manifest.json – cont'd


```
{
"name": "Password",
"uniqueId": "ca.cde.bamboo.endpoint.password",
"displayName": "Bamboo Password",
"type": "password",
"isOptional": false
},
{
"name": "bambooBuildProject",
"uniqueId": "ca.cde.bamboo.endpoint.buildProject",
"displayName": "Bamboo Build Project",
"type": "string",
"isOptional": false
}]
},
```

```
"services": [{
"name": "Run Bamboo Build",
"uniqueId": "ca.cde.bamboo.task.run_build",
"description": "Use this task to run a Bamboo directed build",
"serviceType": "TASK",
"url": "https://hook.io/cde-plugins/runBamboo-build",
"parameters": [{
"planKey": "planKey",
"uniqueId": "ca.cde.bamboo.task.run_build.planKey",
"displayName": "Plan Key",
"type": "string",
"isOptional": false
},
{
"name": "buildStatus",
"uniqueId": "ca.cde.bamboo.task.run_build.buildStatus",
"displayName": "Build Status",
"type": "string",
"isOptional": true
}]
}]
}
```


ca technologies

Online Hosted Plug-in

- You can create an online hosted plug-in
- You can use github and hook.io for these purposes
- Hook.io hosts the micro service that you will be calling
- Github hosts the manifest.json and a copy of the JavaScript
  - Under github, you will need to place these files under the gh-pages branch for them to be accessible

Offline Hosted Plug-in

- This is a web JAVA project under Eclipse or IntelliJ
- Uses the traditional JAVA services and facilities
- Eclipse EE recommended
- It generates a WAR, which will be made part of the tomcat installation

CDE Plug-in Best Practices

- **Portable Manifest**
  - Keep the content of the manifest portable and agnostic to the exact location of the server container
- **Sensitive Information**
  - Store all sensitive information, such as passwords, as endpoint parameters.
    Store the network connectivity details of remote components as endpoint parameters

CDE Plug-in Best Practices – Cont'd

- **Packaging**
  - Ensure the manifest is part of the plug-in package. We do not recommend that you store the manifest separately from the rest of the plug-in.
  - Ensure the manifest resides at the root of the plug-in package.

- **Naming and Versioning**
  - Do not create multiple plug-ins with the same name. Ensure that each unique ID is different
  - Ensure the unique ID of the plug-in is a string that includes the plug-in name and the vendor
  - After you have published the plug-in to the community, do not change the plug-in name and vendor in subsequent versions
  - Update the plug-in version every time you modify and release the plug-in

Why is This Important?

- Creating your own plug-in allows you to integrate your tools with RA CDE
- You can complete the continuous delivery of your releases

Session Summary

- This session presented with the following information
  - You can create your own RA CDE plug-ins based on your business requirements
  - Your plug-ins can either be hosted online or offline
  - This allows you to integrate the different tools that might be available in your enterprise as you are creating the necessary continuous delivery of your releases

Questions?

Comments?

In Closing

Questions after the webcast?

- Post them on the community site
- Contact me directly at walter.guerrero@ca.com

Upcoming RA Community webcasts

- June 29th, 11:00 AM EDT, CA Release Automation V6 Architecture Overview, Keith Puzey

ca technologies

Walter Guerrero
Eng. Svcs Architect
walter.guerrero@ca.com