Big Book of Cucumber

What is this page?

This page collects everything Cambia knows about Cucumber in a single place, along with a few references to outside sources of Cucumber information.

What is Cucumber?

Cucumber is a free, open-source, third-party tool for writing, organizing, and testing software requirements.

It has two parts:

- **Examples (formerly known as Scenarios)** are English-language descriptions of how software should behave
- **Step definitions** are chunks of Ruby or JavaScript code that convert Examples into runnable acceptance tests.

What’s so great about Cucumber?

It helps people in technical and non-technical roles collaboratively write business requirements for our software

When a team uses Cucumber to help build software, they gather all the stakeholders (including product owners, engineers, and business folks) to explore the problem the software will solve, and write Cucumber examples that describe the behavior they want. This discussion reveals misunderstandings and assumptions early in the process, when they’re still cheap to fix.

Every requirement can become an automated test

Once the team has finished writing Cucumber examples, they can optionally write step definitions in Ruby or JavaScript, and sometimes using other libraries such as Selenium or Puppeteer, that turn those examples into automated acceptance tests. As they start to deliver product code, these tests reveal which requirements are implemented and working, and which aren’t done yet.

It creates living documentation that changes as the requirements change

Cambia developed a tool called the “Feature Tree” which displays Cucumber examples in a structured, easy-to-read format that turns examples into a map and description of all the user-facing features in our software. If a product owner or designer changes their mind about how they want the software to behave, the team updates the Cucumber examples and step definitions before changing the product code, ensuring that the documentation is always up to date. Everyone involved with the software (including product owners, developers, SDETs, UX designers, technical writers, managers, business analysts, and customer support) can use the Feature Tree to understand the product’s features.

It stores requirements, automated tests, and documentation in the same GitHub repos as product code

Cucumber examples and step definitions live in GitHub, right next to the product code they’re associated with. This means you no longer have to hunt across Confluence, Slack, SharePoint, Jira, and email to find your software’s requirements, tests, or documentation. They’re always in the same predictable place: right in GitHub, in the same repo as your product code.

It supports Behavior-Driven Development (BDD)

The BDD development philosophy says that you first explain how your application should behave, and then your team writes product code to make it behave that way, and development stops as soon as all the required behavior is implemented. This makes sure your product does exactly what it's supposed to, but no more. Cucumber is a perfect fit for doing BDD.

It tracks how close to “done” our software is

Since all features are captured in Cucumber examples before coding begins, we can use the pass rate of those examples to tell us how close to done the software is. When all Examples pass, we're done.

It identifies bugs or regressions

When new or refactored product code introduces bugs that lead to unexpected or undesired behavior, Cucumber tells us by failing tests. When Cucumber examples are all green, we know the product is behaving the way it's supposed to.

Who uses Cucumber?

Cucumber is required for any Janus microservice or application made up of Janus microservices.

Everyone on a development team that writes Janus microservices or applications built on Janus microservices should become familiar with reading and writing good Cucumber examples, so they can understand the product's intended features and behavior. This includes product owners, developers, SDETs, UX designers, technical writers, and managers.

Everyone on the team who has programming experience should contribute to writing the step definitions that turn the Cucumber examples into runnable tests.

Training resources

Cambia provides several ways to learn more about Cucumber if the *Big Book of Cucumber* that you're reading right now isn't enough:

- Live Cucumber training classes that last from 1 to 3 hours. Schedule through Rick Moerloos.
- Janus Gherkin training video: https://cambiahealth.app.box.com/file/280560057801
- Janus Gherkin training: on the Cambia Learn site
- Janus Essentials training on new Gherkin keywords and best practices for writing Examples: https://web.microsoftstream.com/video/e8fc8489-6549-4e37-9f7e-4ac6c3c3680b
- Ruby training for writing step definitions: basic ruby training on Cambia skillport (after entering skillport, search for "basic ruby training")
- *The Cucumber Book, 2nd Ed.* by Matt Wynne is a good reference. There are several copies in the office.
- Cambia offers subscriptions to the enormous O'Reilly online library, which includes a few Cucumber-related resources. Contact @ Matthew Egbert or Dan Michael for access or more information.

Old vs. new Cucumber keywords

As of November 2020, Cambia recommends transitioning away from 3 old, deprecated Gherkin keywords to 3 new, semantically equivalent keywords for all new Gherkin Scenarios/Examples that your teams write. We also recommend that teams use the automated Gherkin update tool to convert existing Scenarios/Examples to use the new keywords. The tool is available at https://github.com/cambiahealth/gherkin-tool .

The new keywords are *semantically identical* to the old keywords, which means they are drop-in replacements for the old keywords. You don't have to make any other changes to your Scenarios/Examples to make them work with the new keywords.

The deprecated keywords have not been removed from Gherkin and will still work for the foreseeable future. Shifting to the new keywords is strongly advised by Cambia and Cucumber, but not strictly necessary.

Old deprecated keyword	**New replacement keyword**
Scenario	Example
Scenario Outline	Scenario Template
Examples	Scenarios

For example, these Scenarios/Examples use the old keywords:

```
Scenario: foo
    Given a
    When b
    Then c

Scenario Outline: bar
    Given d
    When e
    Then f <NAME>
    Examples:
      | NAME    |
      | Greta   |
      | Hannah  |
```

Here are semantically identical Scenarios/Examples that use the new keywords:

```
Example: foo
  Given a
  When b
  Then c

Scenario Template: bar
    Given d
    When e
    Then f <NAME>
    Scenarios:
        | NAME   |
        | Greta  |
        | Hannah |
```

There's also a new keyword available as of November, 2020: **Rule**. This keyword creates a new organizational layer within feature files.

Without the **Rule** keyword, Scenarios/Examples are collected within a feature file in a flat hierarchy. There's no way to organize the Scenarios /Examples within the file, other than by changing their order within the file:

```
Feature: password management
  Scenario: alpha
  Scenario: beta
  Scenario: gamma
  Scenario: delta
```

The new **Rule** keyword lets you add an extra layer of structure to the feature file:

```
Feature: password management

  Rule: user can change password at any time
    Scenario: alpha
    Scenario: beta

  Rule: user must change expired password
    Scenario: gamma
    Scenario: delta
```

Gherkin doesn't support sub-rules; all rules exist at the same hierarchical layer.

The **Rule** keyword is optional. Existing Scenarios/Examples without **Rule** will continue to work as they always have.

Example mapping

Sometimes a team can successfully brainstorm their way through writing Cucumber Examples. Other times they need the help of an algorithm to follow, to help them jumpstart the Example-writing process, and to help them come up with edge cases they might have missed otherwise. Example mapping is exactly this algorithm. It was designed by the authors of Cucumber, and we've found it to be useful in many situations at Cambia.

These references explain how to use example mapping:

- Cambia-produced training video (11 minutes)
- Cucumber-produced training video (39 minutes)
- Cucumber blog entry

When should I use example mapping?

If your story is **short**, **simple**, and/or **well-understood**, it's probably more efficient for your team to skip example mapping and just write Cucumber Examples by free-form brainstorming.

If your story is **long**, **complicated**, and/or **poorly understood**, your team will probably write more and better Cucumber Examples by relying on the extra structure and step-by-step instructions provided by example mapping.

Example mapping does take more time than free-form brainstorming – it's a heavier-weight process -- so if you're not sure whether to use it for particular Example-writing session, you could start with brainstorming and escalate to example mapping if you're not satisfied with the Examples you came up with.

Best practices for writing Cucumber examples

We developed these best practices while helping Cambia teams write Gherkin Examples. Many of them are also promoted by Cucumber's own training teams and developers.

These are good starting points to follow, but use your judgment to override any best practice that doesn't work well for your particular Example.

General titles, concrete steps

Example titles should use **general language**. Example steps, on the other hand, should include **concrete details or data**. This turns examples into stories: more vivid, more powerful, and more memorable.

Bad

```
# BAD: title has specific details, step lacks specific details

Example: search for doctors near 97201
    When the user searches for doctors by ZIP
```

Good

```
Example: search for doctors by ZIP
    When Jane searches for doctors near 97201
```

Include only relevant details

It's great to include concrete details in your steps, but make sure every detail is relevant to the focus of the Example. Omit unnecessary details.

Bad

```
# BAD: her password has nothing to do with the rest of the Example

Example: add item to cart
  Given Jane's cart is empty
  And her password is "P@ssword"  # <-- omit this step!
  When she adds 1 apple to her cart
  Then her cart shows 1 apple and nothing else
```

Good

```
Example: add item to cart
  Given Jane's cart is empty
  When she adds 1 apple to her cart
  Then her cart shows 1 apple and nothing else
```

Use personas

Personas are fictional test users or other entities (like an employer group) that everyone on the team is familiar with, and that you use as the "actor" in your examples.

Using personas helps to turn examples into vivid, memorable, understandable stories. And when the whole team is familiar with a few personas that are reused across multiple examples, it helps the team think of new examples based on the personas' interests and traits.

Bad

```
# BAD: example doesn't use a persona

When the user searches for a doctor
```

Good

```
When Jane searches for a doctor
```

How do you define a persona? With another Cucumber example! These examples look strange because they contain only `Given` steps, but technically they're still examples.

Persona definition

```
Example: persona definition for Jane
    Given Jane is 51
    And Jane has linked her health plan
    And Jane uses a wheelchair
```

Persona definition examples need step definitions just like any other example. These step definitions can either configure the test user or else verify that the test user is already configured correctly and hasn't "drifted."

This means that persona definition examples are executable tests just like any other example. If a persona definition example fails, that means that other examples that rely on that persona might also fail. Figure out why the persona definition failed and fix that problem, and then re-run all your tests; some of the other failing tests might now pass!

Never use "I" as the subject

Whenever possible, use a persona as the subject of an Example (see section above TODO: ADD LINK HERE). But when it doesn't make sense to use a persona, avoid using "I" as the subject. Use a description of the subject instead: "the user" or "the caller" or another description of the actor.

- **Preferred:** `When Jane searches for doctors`
- **Second choice:** `When the user searches for doctors` or `When the caller searches for doctors`
- **Avoid:** `When I search for doctors`

Avoid technical or implementation details that aren't relevant to user behavior

Focus Examples on user behavior and business rules, not the way in which the code carries out those business rules. Examples consider the service under test to be a black box, and don't describe how the service does its work. Instead, they describe what the user does and what the service does for the user as a result.

Bad

```
# BAD: THEN statement includes implementation details that user doesn't
care about

Example: redirect to log in page
  Given Jane is not logged in
  When she tries to visit her cart directly
  Then the service returns a 302 redirect to the log in page
```

Good

```
Example: service requires identity
  Given Jane is not logged in
  When she tries to visit her cart directly
  Then she sees the log in page instead
```

Limit the number of Examples

Examples should cover happy paths and a handful of the most critical unhappy paths.

There's no need to capture every input combination or workflow permutation in examples: put those in unit tests. Teams are easily swamped by the effort of writing and maintaining too many Examples if they try to cover every possible case.

Limit the scope of each Example

Whenever possible use only one `When` statement per Example. If you have lots of `When` or `And` statements under a `When`, you're probably doing too much in that Example.

This keeps Examples focused. When an Example fails, it's easier to tell what part of the feature is broken. It's also easier for developers to understand and code against tightly focused Examples.

Bad

```
# BAD: 1 Example covers 2 different features

Example: log in and search for doctors
    When Jane logs in
    And she searches for doctors near 97201
    Then ...
```

Good

```
# Each separate feature gets its own dedicated Example

Example: log in
  ...

Example: search for doctors by ZIP
  ...
```

Another way to limit the scope of Examples is to combine or eliminate steps when possible. You can often imply behavior or states of the world without mentioning them explicitly.

Bad

```
# BAD: too many steps and details

Given Jane is not logged in
And she logs in with her name and password
...
```

Good

```
Given Jane is logged in
```

Use `Given` steps only when needed

The standard Cucumber Example that you learn about in training has 1 `Given`step, 1 `When` step, and 1 `Then` step. But in the real world, you can often omit the `Given` step.

Remember that `Given` steps express the state of the system before the `When` step applies. In many cases there is no particular state of the system that needs to be specified or set up, so you don't need a `Given`.

Good

```
# No Given step needed

When an anonymous caller hits the unprotected API endpoint asking for
doctors in 97201
Then the system returns only doctors in 97201
```

Use Scenario Templates only under certain conditions

Cucumber's Scenario Template feature is a powerful way to make examples easier to understand and more succinct with no loss of clarity. But this feature should only be in certain circumstances.

Avoid Scenario Templates when each row in the Scenarios table produces *different* expected behavior.

Do use Scenario Templates when each row has *identical* expected behavior (that is, each row has identical `Then` steps).

Bad

```
# BAD: each row in table has different expected behavior,
# so using a Scenario Template is not a good idea.

Scenario Template: search by address
    When Jane searches for doctors near <ADDRESS>
    Then the address is treated as a <DESCRIPTION>
    Scenarios:
        | ADDRESS     | DESCRIPTION          |
        | 10 Main St. | well-formed address  |
        | 10          | malformed address    |
        | -7 Main St. | non-existent address |
```

Good

```
# These addresses should all be handled in the same way,
# so using a Scenario Template is a good idea.

Scenario Template: search by address
    When Jane searches for doctors near <ADDRESS>
    Then she sees doctors near that address
    Scenarios:
        | ADDRESS        |
        | 10 Main St.    |
        | 10 Main St     |
        | 10 Main st.    |
        | 10 Main st     |
        | 10 Main Street |
        | 10 Main street |

# These two error cases should be handled differently from each other

Example: search by malformed address
Example: search by non-existent address
```

Good

```
# These protocols should be handled in the same way,
# so using a Scenario Template is a good idea.

Scenario Template: import claims data
    When the system imports claims via <PROTOCOL>
    Then the claims appear
    Scenarios:
        | PROTOCOL |
        | rest     |
        | soap     |
```

Use `When` and `Then` correctly

Use `When` for actions, not setup. Use `Then` for results, not actions.

Teams sometimes use `When` where they should use `Given`, and `Then` where they should use `When`.

Bad

```
# BAD: WHEN is used here for setup, THEN is used here for action

Example: log in
    When Jane has an account
    Then she can log in
```

Good

```
Example: log in
    Given Jane has an account
    When she logs in
    Then she can access any page
```

Write Examples iteratively

You almost never get an Example exactly right on the first pass. Expect to rewrite it several times, and budget time for this. There are 3 times when it often makes sense to rewrite Examples:

- After getting feedback on the Examples from stakeholders
- While writing step definitions for the Examples
- While developing product code that the Examples describe

Describe features, not controls

Examples should talk about the user's intentions instead of the GUI controls they have to use: describe what the user is trying to do, not how they are doing it. Another way to put this is that Examples should focus on requirements and business rules, not implementation details.

Bad

```
# BAD: steps focus on GUI controls

When Jane enters "97201" in the ZIP field
And she clicks the "search" button
```

Good

```
When Jane searches for doctors near 97201
```

Use present verb tense everywhere

You might be tempted to use past tense for GIVEN steps, present tense for WHEN steps, and future tense for THEN steps. This is how many of us naturally write Examples if we're not paying attention to our tenses.

But Examples are easier to read and understand when they use present tense for all GIVEN, WHEN, and THEN steps.

Bad

```
# BAD: each step uses a different verb tense

Given Jane has logged in
When she searches for female doctors
Then she will see only female doctors
```

Good

```
Given Jane logs in
When she searches for female doctors
Then she sees only female doctors
```

Write like you talk

Conversational Gherkin is easier to understand and natutrally focuses on the right level of abstraction (i.e., the user's intention, not the GUI controls). Read each Example out loud and see it sounds like something you'd say to a colleague over lunch while describing your software's features.

Bad

```
# BAD: too verbose & not how people talk

When Jane searches for a list of doctors of only the female variety
```

Bad

```
# BAD: too short & not how people talk

When female doctor search
```

Good

```
When Jane searches for female doctors
```

Follow normal English conventions for writing

Pay attention to your spelling, capitalization, and grammar. Make your high school English teacher proud.

One exception: don't use periods at the end of Feature titles, Example titles, or steps. The only place you should use periods is at the end of free English text in the Description field.

Bad

```
# BAD: periods where they shouldn't be,
#      no period at end of description field,
#      bad grammar in steps,
#      non-standard capitalization in steps

Feature: Password management.

    Example: Change password.

    For detailed rules on password requirements, see our security PDF

        When Jane change her PASSWORD.
        Then she can logs in With the new password.
```

Good

```
Feature: Password management

    Example: Change password

    For detailed rules on password requirements, see our security PDF.

        When Jane changes her password
        Then she can log in with the new password
```

Use the active voice

Using the active voice means including an actor and an action that the actor is performing, instead of saying that some action was performed. For example: “I threw the ball” uses the active voice, while “the ball was thrown” uses the passive voice.

Using the active voice whenever possible makes Examples more direct, more vivid, and more story-like. It also encourages the use of personas, since the active voice requires an actor.

Bad

```
BAD: written in passive voice

When psychiatrists are searched for
```

Good

```
When Jane searches for psychiatrists
```

Use the `Background` keyword

Extracting common `Given` steps into a `Background` makes Examples more succinct and easier to maintain.

Bad

```
# BAD: both Examples have identical GIVEN steps

Example: foo
    Given a
    When b
    Then c

Example: bar
    Given a
    When d
    Then e
```

Good

```
Background:
    Given a

Scenario: foo
    When b
    Then c

Scenario: bar
    When d
    Then e
```

Avoid “should” or “must”

Omitting “should” or “must” in `Then` statements makes Examples shorter, more direct, and more confident. “Should” or “must” is implied anyway: if the system doesn’t behave like it should, the test will fail.

Bad

```
# BAD: steps include "should" or "must"

Then the results should include "Dr. Baker"
Then the results must include "Dr. Baker"
```

Good

```
THEN the results include "Dr. Baker"
```

Keep Example titles short

Titles should describe as briefly as possible what the user is trying to do. Details belong in the steps, not the title.

Bad

```
# BAD: title is too long and has unnecessary details

Example: searching for doctors by ZIP returns a list of doctors
         that are close to that ZIP code
```

Good

```
Example: search for doctors by ZIP
```

Use And steps sparingly

Sometimes you need to use an `And` step because there are two completely separate things to check, which could each fail independently. This is especially common with `Then` steps:

```
Then Jane is redirected to the checkout page
And she sees the sales tax for the items in her cart
```

But if it makes sense to combine an `And` step with the step before it, it’s often a good idea to do so. This is frequently true of `Given` steps:

```
# BAD: unnecessary "And" step

Given Jane is logged in
And she is a Cambia PPO user


# GOOD: combine the "Given" and "And" steps

Given Jane is a logged-in Cambia PPO user
```

Don’t refer to customers

If you’re writing Examples that will be read by actual or potential customers (for example, if you work for HealthSparq, Blue Cross organizations from other states might read your Examples when deciding whether to buy HealthSparq services) then your Examples shouldn’t refer to customers by name. Instead, refer to features of your software that customers might be interested in.

For example, imagine that Blue Cross of Michigan is interested in HealthSparq’s “cost” feature:

Bad

```
# BAD: refers to a customer

Given Jane is a logged-in Michigan user
```

Good

```
Given Jane is a logged-in user
And the cost feature is enabled
```

Non-functional requirements

You can think of non-functional requirements as requirements that aren’t user-facing features. For example, performance or security requirements are usually considered non-functional. In many cases you can capture non-functional requirements in Examples, just like you do with functional requirements.

If there are better ways for your team to capture non-functional requirements, you are not required to use Cucumber Examples.

Good

```
# Performance-related non-functional requirement

Example: support multiple simultaneous login attempts
  When 30 users try to log in within a 1-second span
  Then all 30 users successfully log in within 5 seconds


# Security-related non-functional requirement

Example: guard against SQL injection attacks
  When Jane logs in using "105; DROP TABLE Users" as her username
  Then she sees a warning that her username is invalid
  And the Users database table is not affected
```

Tags

You can add tags to any Cucumber Example or Feature. To add the tag `@foo` to an entire feature (i.e., that tag applies to **all** the scenarios within the feature file) and the tag `@bar` to a single Example, use this syntax:

```
@foo
Feature: password management

    @bar
    Example: password expires
```

There are two ways you can use tags with Cucumber:

- Execute **only** Examples that are labelled with one or more particular tags.
- Execute all Examples **except** those that are labelled with one or more particular tags.

For example, you could save time by adding a temporary `@wip` tag to all Examples you're currently writing step definitions for, and tell Cucumber to execute only Examples with that tag.

Or you could ask Cucumber to run all Examples except for ones that take a long time to complete by excluding Examples that have the `@slow` tag.

Janus-approved tags

Although Cucumber will let you use any arbitrary tag, we want to keep the number of tags in use at Cambia from growing too big. Imagine the confusion if there were 20 different tags that all meant "this test runs slowly, so should only be run on weekends." So try to only use one of the tags in the "Janus-approved" list below. If your team wants to propose a new tag, consult with a member of the Quality team.

It's fine to **temporarily** use any tag you want, as long as you intend to remove it from your Examples within a week or so.

Approved tag	Meaning
`@SafeForIntegration`	Run this Example during pre-deployment, and again against deployed services in an integration environment. See TODO CHANGE THIS TO LINK TO THE RIGHT SECTION BELOW https://cambiahealth.atlassian.net/wiki/display/JANUS/Cucumber%3A+guidelines+for+running+post-deployment+tests

`@IntegrationOnly`	Run this Example against a deployed service in an integration environment. See TODO CHANGE THIS TO LINK TO THE RIGHT SECTION BELOWhttps://cambiahealth.atlassian.net/wiki/display/JANUS/Cucumber%3A+guidelines+for+running+post-deployment+tests
`@Manual`	A valid business requirement that can't be tested in an automated way. This Example should be tested manually instead.
`@Bug`	Cucumber should **not** run this Example: it is expected to fail because of a known bug in the product code. See the Workflow for failing tests section of this document.

Feature Tree

Cucumber Examples are specifications and executable tests, but they're also living documentation! Since each dev team must keep its Examples up to date as its product behavior changes (if they don't, their tests fail and their CI/CD pipeline halts), those Cucumber Examples give an accurate view of how the software is expected to behave when it's fully delivered by the team. And if you look at which Cucumber Examples are passing and which are failing, you get an accurate view of how much of that product's functionality has actually been delivered by the dev team so far.

The best way to view a product's Cucumber Examples and the pass/fail status of those Examples is with Cambia's **Feature Tree**. Access the Feature Tree for any application or microservice through the Minerva dashboard.

To learn more about the benefits of the Feature Tree and the living documentation provided by Cucumber Examples, see this presentation:

How do I make my service's Examples show up in the Feature Tree?

First step: add `cucumber-orb` to your pipeline configuration (CircleCI configuration file). This orb creates the cucumber files that Sunomono, our test runner, uses to kick off Feature Coverage Generator, which is what creates the Feature Tree for your service. More information on creating a CircleCI configuration file is here.

Even if you have only manual Examples, or known bugs that mean some of your Cucumber tests don't pass, you should still have Cucumber tests enabled by including `cucumber-orb` in your pipeline configuration. Add `@Manual` and/or `@Bug` tags to any Examples that you'd like Cucumber not to run. More information on tags, see above.

If your pipeline configuration is set up properly and your Cucumber test run is finishing successfully in the pipeline but you're still not seeing a Feature Tree with your service's Examples, contact our team or file a ticket: https://cambiahealth.atlassian.net/wiki/spaces/JANUS/pages/364021845/HowTo+Create+Janus+Quality+Support+Ticket?moved=true.

Why does my multi-service/domain Feature Tree not include all the services I expect?

Minerva maintains the lists of what services belong to a domain, so if your multi-service report doesn't show all the services you expect, contact a member of the Minerva team (the `#minerva` Slack channel is a good way) or submit a PR to update the Minerva file that contains the lists of services. Submit changes to the `services.clj` file in the Minerva repo.

File names and directory structure

Cucumber has conventions for naming files and directories. The directory structure of a Cucumber project must look like this:

```
.
 features
     step_definitions
        alpha_steps.rb
        beta_steps.rb
     sub_feature_one
        foo.feature
        bar.feature
     sub_feature_two
        baz.feature
        qux.feature
     support
         env.rb
```

Feature files belong in the `features/` directory or one of its sub-directories.

All files and directories should be spelled using `snake_case` instead of `camelCase` or `kebab-case`.

Step definition filenames should end with `_steps.rb`, such as `reward_redemption_steps.rb`.

You can use Cucumber to create this standard directory structure for you:

```
% cucumber --init
  create   features
  create   features/step_definitions
  create   features/support
  create   features/support/env.rb
```

Guidelines for running post-deployment tests

Running a handful of user acceptance tests post-deployment gives teams peace of mind knowing that their deployed stack is working as expected.

In Janus, the bulk of test automation is focused on pre-deployment verification. But the pre-deployment environment in which Cucumber tests are run is different from the environment that a stack is deployed to, so it's possible for code to pass all tests in a pre-deployment environment but fail in a production environment. This is why post-deployment tests are useful.

Strategy

You can ensure that a Cucumber Example runs in a post-deployment environment by adding the `@SafeForIntegration` tag to that Example. All Examples with this tag will run after each deployment of the stack in the Dev environment only.

Add the `@IntegrationOnly` tag to Examples that should run in a post-deployment environment but for some reason should **not** run in a pre-deployment environment.

Guidelines

Which examples should get this tag?

Consider the following when selecting which Examples to tag with `@SafeForIntegration`.

Ideally, any Example with this tag should have no side effects. This means it should satisfy both of these conditions:

- It should not require any specific state of the system
- It should not change the state of the system, either within the stack under test (e.g., no records modified in the database) or within any external dependencies (e.g., other Janus stacks or 3rd party services).

This is a tough constraint to satisfy, and many stacks won't have any Examples that qualify. In that case, your team can loosen the criteria to include Examples that modify state only within the stack under test (i.e., they must not propagate state changes outside of the stack). However, it's **critical** that the Example revert these state changes after it completes, even in cases where the Example fails!

Additional considerations:

- Examples that depend on a mock server (e.g., to verify that the stack under test called out to an external service) cannot currently run successfully in a post-deployment environment (although there may be infrastructure changes in the future to support this).
- Because Examples with the `@SafeForIntegration` tag should ideally have no side effects, they will automatically satisfy two criteria that should apply to all Examples, regardless of what environment they run in:
  - **Idempotent**: you can run the same Example successfully multiple times in a row
  - **Independent**: you can run the Examples in any order without affecting their outcome [TODO: MAKE SEPARATE SECTION IN THIS DOC REPEATING THIS]
- As mentioned above, if an Example does change state, the Example must revert all of those state changes, whether or not it successfully gets through all of its steps. To make sure this happens, inspect each of step in the Example for possible state changes and include the necessary logic and guard rails to back out those changes no matter what happens within each step. For example, if a `Given` step makes state changes and then fails before the Example moves on to its `When` or `Then` steps, the `Given` step must undo its work.
- Some services require runtime configuration unique to execution of Cucumber Examples (e.g., authentication-service has runtime configuration that defines tenants differently in different environments).

Implementation patterns

env.rb

- Pre-deployment and post-deployment execution workflows store the endpoint of the stack under test in the environment variable `SERVICE_ENDPOINT`. It's convenient for local debugging to default to the localhost port that your stack runs on.

```
endpoint = ENV['SERVICE_ENDPOINT'] || 'http://localhost:8080'
```

- Any code that interacts with `docker-compose-tests.yml` dependencies should be behind a conditional that only runs when `JANUS_ENVRIONMENT` is set to `local` or is unset. When executing in a post-deploy environment, none of these resources will be available, causing Cucumber Examples to fail.

hooks.rb

- Make sure that any interaction with mock servers is behind a conditional which only allows that interaction if `JANUS_ENVIRONMENT` is set to `local` or is unset:

```
After do |scenario|
  # Only clean up mock servers if running locally
  # (mock servers are currently unavailable in post-deployment env)
  if ENV['JANUS_ENVIRONMENT'] == 'local'
    event_logging_service_api.delete('/__admin/requests')
  end
end
```

Dockerfile.test

- Any service's unit tests or integration tests being executed on `cucumber-base` must be behind a conditional that only runs when `JANUS_ENVRIONMENT` is set to `local` or is unset. These tests should not run in a post-deploy environment.

Workflow for failing tests

Tagging an Example as a bug

If an Example is expected to fail because of a product bug, do two things to mark that Example:

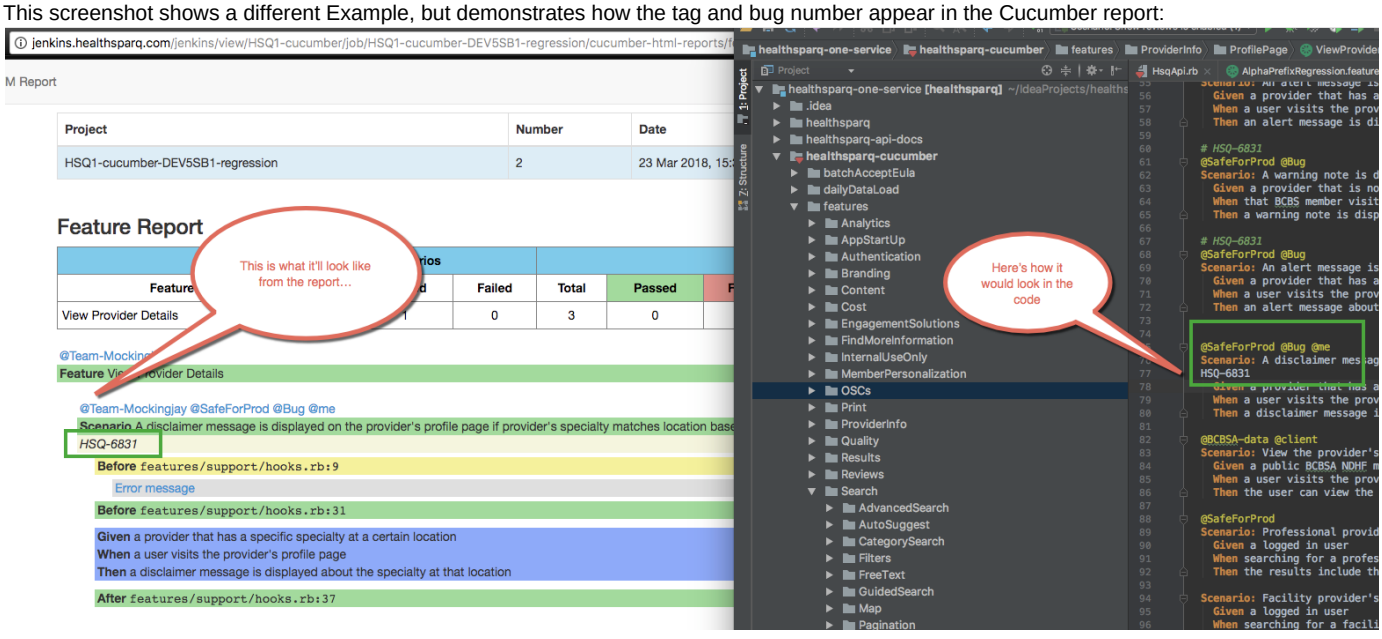
1. Add a @Bug tag immediately above it
2. Add the Jira ticket number in the Example's description field (i.e., between its title and its first step).

For example:

```
Feature: provider search

  @Bug
  Example: search by language

  HSQ-1234

    Given Jane is logged in
    When she searches for providers who speak French
    Then the results include only providers who speak French
```

The @Bug tag prevents Cucumber from running the Example. It also marks the Example as `pending` in Cucumber's HTML report, and includes the bug number in the report.

This screenshot shows a different Example, but demonstrates how the tag and bug number appear in the Cucumber report:



Pending steps

When writing step definitions for an Example, sometimes you will want to use the `pending` method as the only logic in the step definition:

```
# step definition in Ruby

Given(/^(.+) is logged in$/) do |user_name|
  pending
end
```

Use the `pending` method in any step definition that's for an Example that stakeholders have approved for future development, but which you haven't completed the step definition logic for. If the Example hasn't been approved, don't write any step definitions at all for it.

This table shows what should happen with the step definitions at each lifecycle stage of an Example:

Status of Example	Status of step definitions for that Example
When the 3 amigos finish writing the Example…	…step definitions for that example should not yet exist.
When the stakeholders finish reviewing the Example and approve it for future development…	…someone should write step definitions for the Example, using only the `pending` method.
When the dev team assigns the Example to a future sprint…	…someone should flesh out every step definition needed by that Example by replacing the `pending` method with complete logic.
When the dev team starts working on the product code to satisfy the Example…	…step definitions for that Example should already be complete and the Example should fail.
When the dev team finishes the product code to satisfy the Example…	…the Example should pass.

Exit codes in the Janus Jenkins Pipeline

Important! This section is mostly obsolete since Cambia has moved from Jenkins to CircleCI as our build tool. Some members of the Quality group have recommended we keep this information here for reference.

"Cucumber tests exited with code 13"

13 is an exit code set by stack-build to signal that the cucumber-tests container exited with a non-zero exit code.

Look for this line to find the cucumber-tests exit code:

> *"ERROR --- Tests Failed - exit code:"*

In general (for codes other than 137 and 143), the exit code you find in this line will be the exit code from the cucumber-tests container. This is because it comes from the `timeout` command, which forwards the exit code from a `docker wait` command, which forwards the exit code from the Docker container that failed, which is the cucumber-tests container.

Exit Codes

Code	Explanation	Notes
0	Cucumber success	Because output is suppressed for successes you won't actually see an exit code of 0 in the logs. Instead you'll see: "SUCCESS - Tests Passed" and "Cucumber tests complete"
1	Cucumber (Examples) failure	Cucumber tests (Examples) have failed. Check that they pass when you run `janus cuke` locally. However, sometimes you can still get this error in the pipeline when local Cucumber passes, in which case it may be due to a pipeline issue or differences between your local environment and the pipeline environment.

2	Cucumber (run) failure	This exit code 2 should be the code from the failed cucumber-tests container. Cucumber exits with code 2 when the application failed (versus the Example failing). This could be caused by Cucumber exiting or erroring out before the Examples finish running. See: https://github.com/cucumber/cucumber-ruby/pull/845
137	Docker out of memory	This error is due to the Docker container running out of memory.
143	1 hour timeout	Stack-build kills the docker container/process after one hour.

Useful code on how the pipeline sets/treats exit codes: https://git.healthsparq.net/projects/JP/repos/stack-build/browse/scripts/cucumber-tests.sh#358

Also, this might be helpful: https://mgmt2jenkins01.healthsparq.com/failure-cause-management/