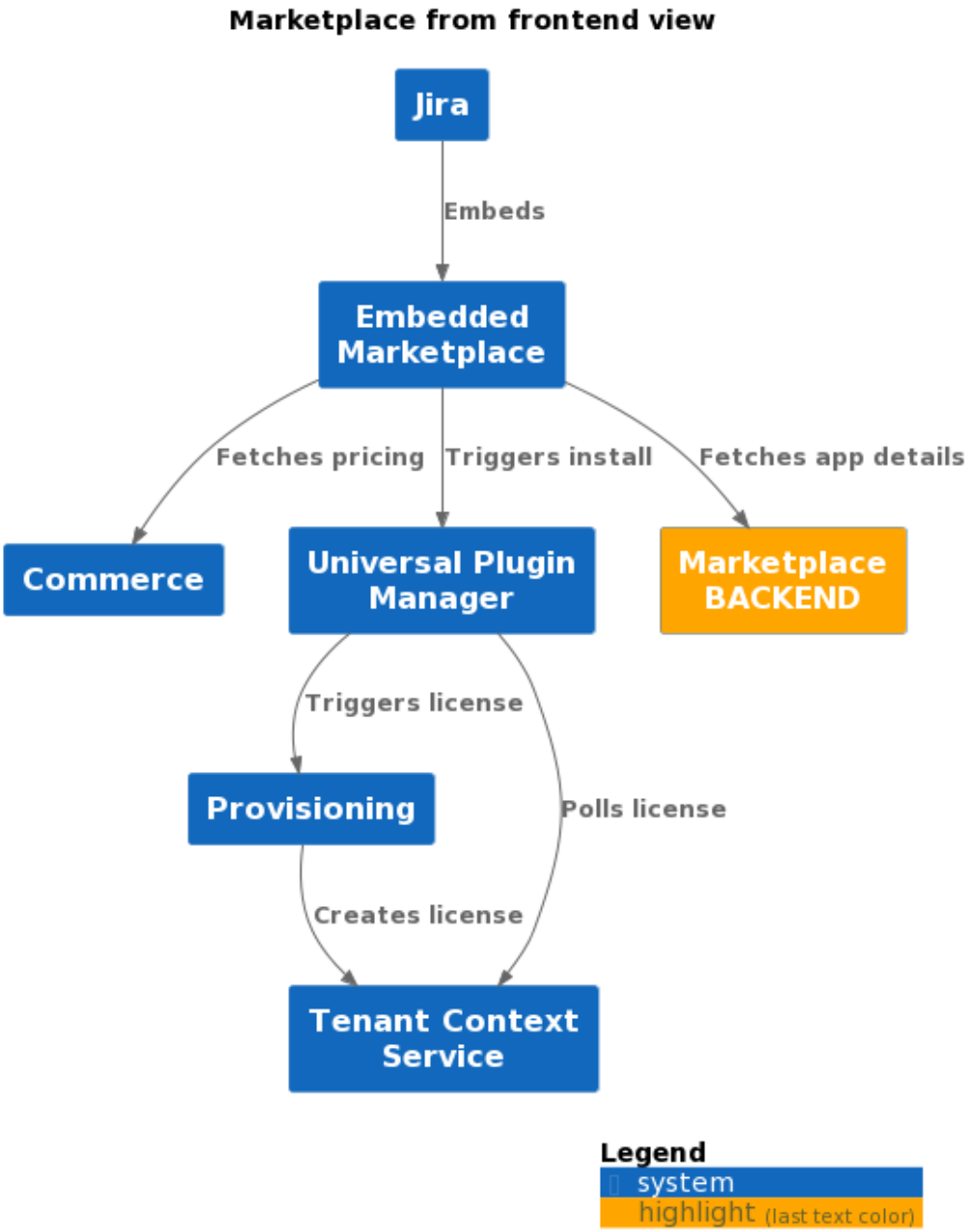
Architecture diagrams should be code

Brian McKenna — 2023-01-09

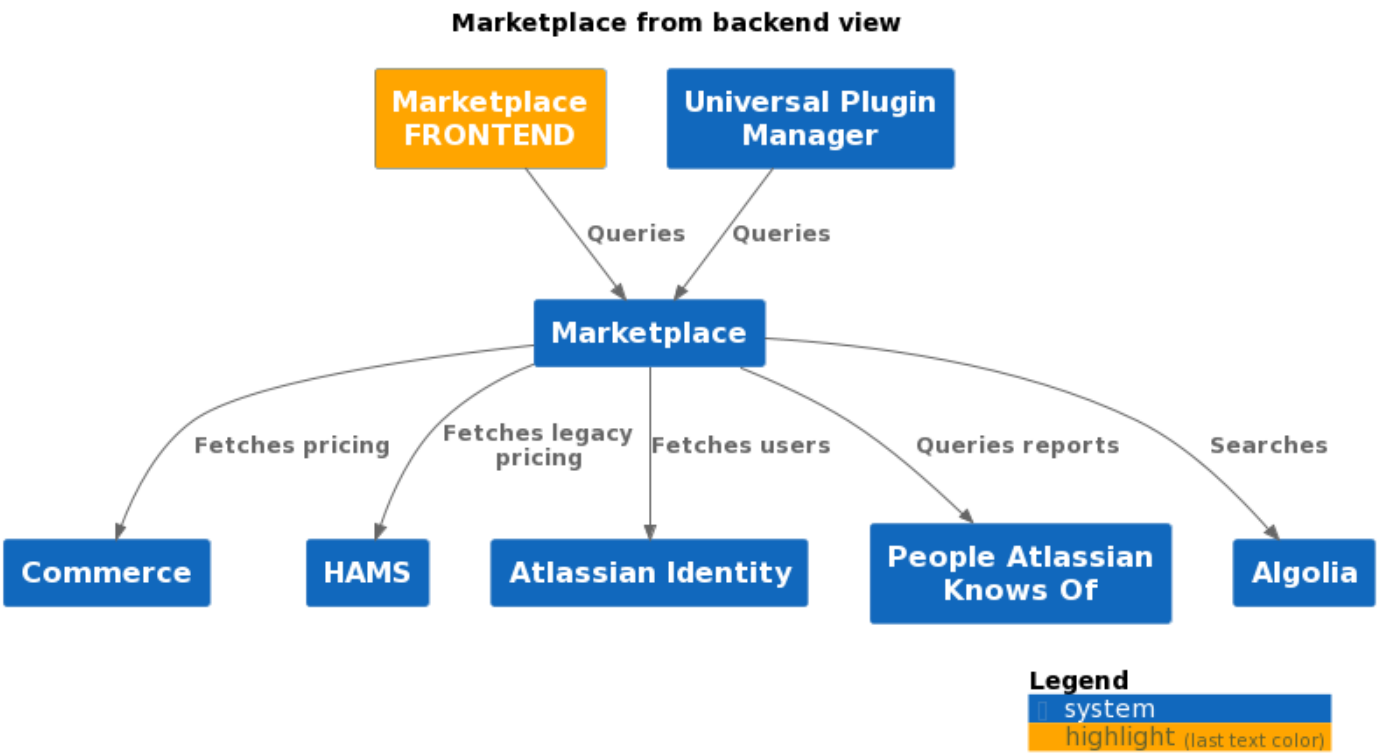
For the past few years I've been the most senior developer on my teams in Atlassian, in both position (Principal Engineer) and time (almost 9 *years*) - this means I usually take on the responsibility of managing our software architecture.

Architecture is the relationships between systems, which can be fairly tricky to talk about. Probably the best form of communication is a diagram, with boxes representing systems (or components) and lines representing relationships between them. This can still have issues.

When my previous engineering manager joined the Atlassian Marketplace team, he asked everyone to draw an architecture diagram. Each came out extremely different. The people focused on frontend had things like Jira, Embedded Marketplace, Commerce, Provisioning, Atlassian Connect, then an arrow to a huge box - just labelled *Marketplace backend*.



The people focused on backend had the reverse - a huge box just labelled *frontend*!



Actual relationships do exist between systems (e.g. network calls, shared storage) but an architecture diagram can't give all details without becoming the code it's meant to represent. That means all architecture diagrams are *views* into an abstraction. It's not wrong to have a big box representing your backend; it's just one way of viewing an abstraction.

Some teams and projects will have a diagram and point to it as "here is *the* architecture" - but it's not *the* architecture, it's just one particular view.

Ideally, we'd have dozens of architecture diagrams - from various views, from various proposals, from various teams. We don't see this very much and I think part of the problem is that architecture diagrams can be costly.

Most diagramming tools require you to use the mouse, pointing and clicking and dragging and drawing. They can be very fiddly and even buggy; e.g. there's one architecture diagram for Atlassian Marketplace with an old outdated box that can't be deleted using the software it was diagrammed in, it just won't allow that, for some reason.

If someone adds a relationship to a new system, will they even remember to visit the Confluence page to click and drag and draw over the architecture diagrams?

Maybe we should write architecture diagrams using code instead. With code, we can update architecture diagrams within a pull request, version them and quickly modify many of them at once.

We can also add some formality. Instead of modelling things with lines and boxes, we can directly model systems and relationships.

The C4 Model formalises architecture into 4 abstractions: system context, containers, components and code. I only use system context and containers; components and code are too detailed. A container is something which runs, such as an application server, database, filesystem, etc. A system represents some group of software, and it can have containers which that software is deployed within.

There are a bunch of tools for using C4. There are two which you might have already used for other things, such as PlantUML and Mermaid. One of the above images was generated from this PlantUML code:

```
@startuml
!include  https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Containe

title Marketplace from backend view

AddElementTag("highlight", $bgColor="orange")

System(marketplace, "Marketplace")
System(commerce, "Commerce")
System(hams, "HAMS")
System(identity, "Atlassian Identity")
System(pako, "People Atlassian Knows Of")
System(frontend, "Marketplace FRONTEND", $tags="highlight")
System(algolia, "Algolia")
System(upm, "Universal Plugin Manager")

Rel(marketplace, commerce, "Fetches pricing")
Rel(marketplace, hams, "Fetches legacy pricing")
Rel(marketplace, identity, "Fetches users")
Rel(marketplace, pako, "Queries reports")
Rel(marketplace, algolia, "Searches")
Rel(frontend, marketplace, "Queries")
Rel(upm, marketplace, "Queries")

SHOW_LEGEND()

@enduml
```

But instead of writing PlantUML, I use Haskell to model architecture. A simple example would look like:

```haskell
data AtlassianSystem
  = Marketplace
  | Identity
  | Pako
  deriving (Eq, Ord, Show)

data AtlassianTechnology
  = REST
  | SQL
  deriving (Eq, Ord, Show)
```

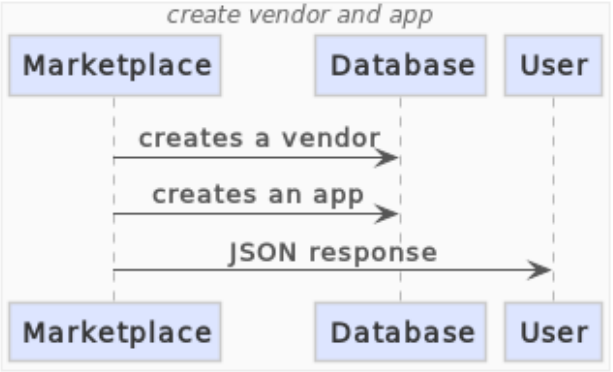
```
marketplaceSystem ::
  SoftwareSystem
    AtlassianSystem
    ()
    AtlassianTechnology
    ()
marketplaceSystem =
  softwareSystem
    & relationships . at Identity ?~ relationship REST
    & relationships . at Pako ?~ relationship SQL

marketplace ::
  Model
    ()
    ()
    AtlassianSystem
    ()
    AtlassianTechnology
    ()
marketplace =
  model
    & softwareSystems . at Marketplace ?~ marketplaceSystem
    & softwareSystems . at Identity ?~ softwareSystem
    & softwareSystems . at Pako ?~ softwareSystem
```

And the Haskell library can spit out PlantUML from this model. There's a few checks which can be implemented on this model, e.g. do any components reference systems which their container doesn't?

Going further into code, most systems have a consistent way of communicating with services. Atlassian has a Service Proxy to handle common service to service communication. When configuring Service Proxy, you give it a list of services that your service depends on. We can easily write some Haskell which takes that configuration and spits out a C4 Model, allowing us to create a bunch of diagrams.

We can also generate a model from running tests. Each test can generate a sequence diagram using something like DataFlow. Collecting the systems used in all tests can generate a C4 system context and container diagram for our whole system. Here's an example sequence diagram generated by running a single test:



I'm pretty convinced that we should be using code to generate architecture diagrams.

Code allows making quick changes. Architecture code can be versioned with the code which implements it. We can write algorithms to check our architecture. I'd like to see more tools available to describe architecture as part of code, allowing us to generate as many diagrams as we want, for accurate and easy communication.

Do you use code to diagram architecture? Do you do something other than just write PlantUML or Mermaid, by hand? I'd love to hear more!

Pieter-Jan Savat

2 years ago

Because the diagrams never seemed to reflect reality (out of date, incorrect, different names, etc) and way too time consuming to keep all details up to date especially with hundreds of microservices, I decided that diagrams should be based on the code directly, as living documentation/diagrams.
So I created www.contextmap.io

0 0 Reply

Subscribe Privacy Do Not Sell My Data

I am Brian McKenna, principal engineer at Atlassian.

My interests cross:

- Programming languages
- Functional programming
- Type theory
- Logic
- Web applications

You can find me on Mastodon.