

Visualising software architecture

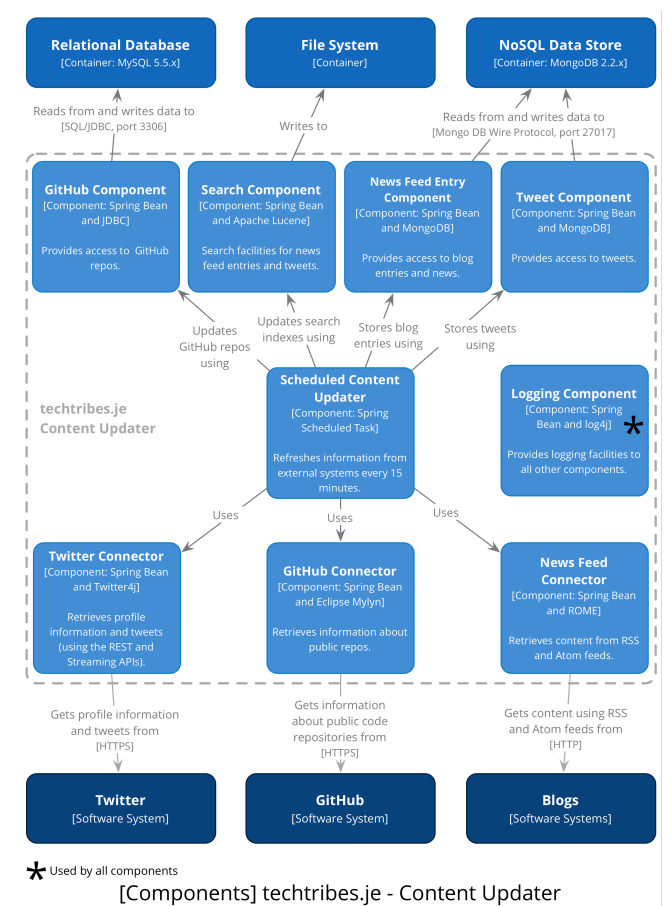
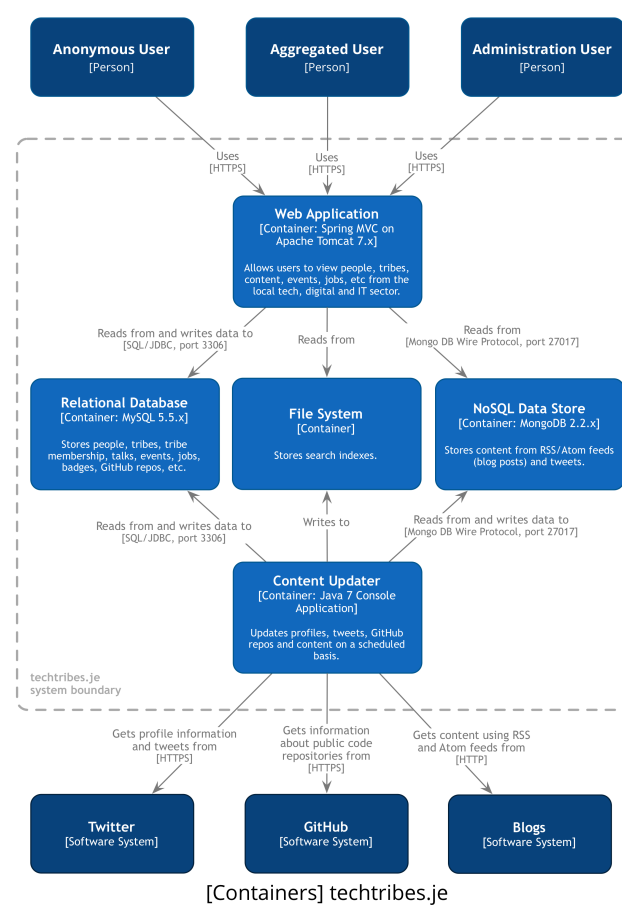
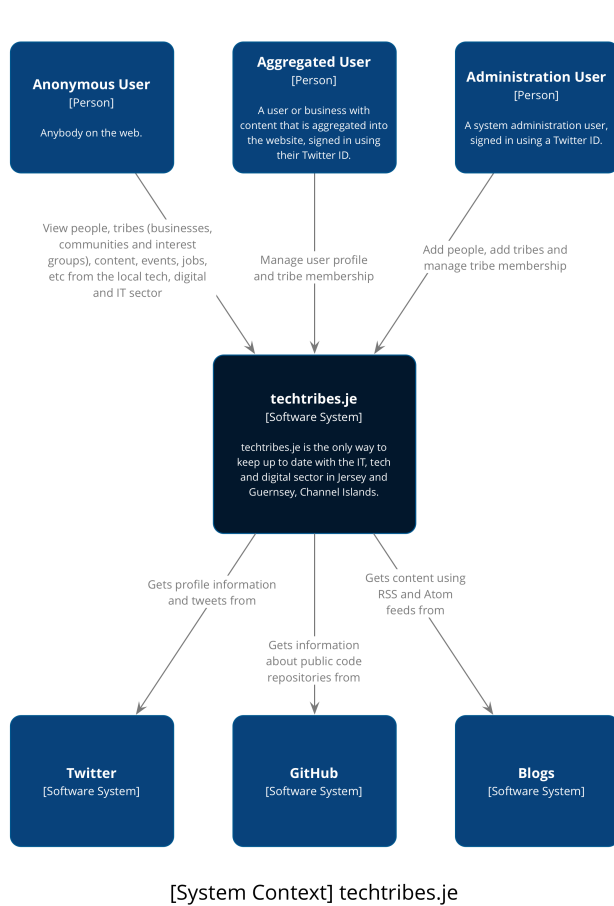
1

As a team, agree upon a set of abstractions you will use to communicate software architecture. The "C4 model" is a hierarchical way to think about the static structures of a software system in terms of containers, components and classes (or code):

A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **classes** (or **code**).

2

Visualise this hierarchy by creating a collection of System Context, Container, Component and (optionally) UML class diagrams. Think about these diagrams as maps of your software, showing different levels of detail.



Level 1: System Context

A System Context diagram is a good starting point for diagramming and documenting a software system, allowing you to step back and see the big picture. Draw a diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with. Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details.

Level 2: Containers

The next step is to illustrate the high-level technology choices with a Container diagram. A "container" is something like a web application, mobile app, desktop application, database, file system, etc. Essentially, a container is a separately deployable unit that executes code or stores data. The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.

Level 3: Components

Next you can zoom in to each container further to visualise the major structural building blocks and their interactions. The Component diagram shows how a container is made up of a number of components, what each of those components are, their responsibilities and the technology/implementation details. If your components don't all fit on a single diagram, create multiple versions showing different portions of the container.

3

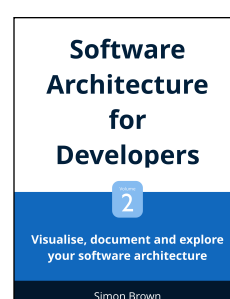
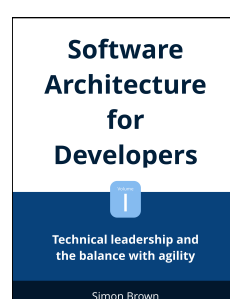
A common set of abstractions is more important than a common notation, but do ensure that your notation (shapes, colours, line styles, acronyms, etc) makes sense. If in doubt, add a diagram key/legend, even when using UML.

4

Use the elements in your model of the static structure to create additional supplementary diagrams in order to communicate runtime behaviour and deployment (the mapping of containers to infrastructure).

"Software Architecture for Developers"

A developer-friendly, practical and pragmatic guide to lightweight software architecture, technical leadership and the balance with agility.



Structurizr
Visualise, document and explore your software architecture

A collection of tooling to help you visualise, document and explore your software architecture.