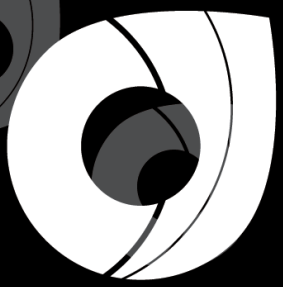
Corndel DevOps Engineering Programme

in association with Softwire

Module 6: Project Exercise



Corndel Digital.

DB1S

Module 6

Project Exercise Brief

In this exercise, you'll create architecture diagrams to describe your todo application. You'll explore the C4 model for creating diagrams at different levels of abstraction, and learn how these diagrams form part of software documentation.

Your diagrams should describe your ToDo app as configured in your production docker container. In later modules we'll revisit and expand these diagrams, keeping them up-to-date as you continue to add functionality.

Architecture Diagrams

Architecture diagrams are a popular way of communicating software architecture. By looking at a diagram, a new developer can quickly understand how the system works, without getting bogged down in implementation details. Architecture diagrams are central to good technical documentation. They're also important when designing new systems and applications.

There are many ways to draw architecture diagrams. Diagrams may differ in their style and notation, and in their level of abstraction. Have a look online and you'll quickly find a diversity of diagram styles. Despite the variation, architecture diagrams all aim to answer two key questions:

- What components is the system built from?
- How do these components interact with each other?

The C4 Model

The C4 Model is one approach to creating useful architecture diagrams. It encourages you to create diagrams at four levels of abstraction:

- Context
- Container
- Component
- Code (optional)

Take a few minutes to read about the C4 Model at https://c4model.com/. There's no need to concern yourself with "Supplementary Diagrams" at this stage - stick with the core four.

The C4 Model, diagram is defined by its *level of abstraction*, rather than any specific style or notation. The highest level of abstraction (Context) shows how the software fits into a larger architecture, and each subsequent diagram zooms in to your app, providing more detail on a specific part.

Exercise

Part 1: Context Diagram

It's normal to start with a high-level diagram of your system. High-level diagrams are simple to create and understand. They also provide useful context for more detailed low-level diagrams.

Create a Context diagram for your todo app. A context diagram simplifies your application into a single box, and focuses on its *context* - the relationships between the application, users, and external systems. You can draw the diagram using whatever notation you find useful - boxes and arrows are conventional.

Be sure to annotate your diagram. It should be clear what each box and arrow represents.

Use a digital drawing tool to create the diagram. The tool itself doesn't matter too much: https://app.diagrams.net/ is one good option.

Hints:

- The context diagram should show your application's *context*, not any internal details.
- Your app's context should include the user!
- Is Trello part of the context, or an internal detail?

Part 2: Container Diagram

Add a second diagram, this time focusing on the Container level. A container diagram illustrates the key internal components of your app, and how they interact with the user. Your container diagram should include, for example, the relationships between gunicorn, flask and your application code.

Submitting your work

Although we aren't writing code for this exercise, we will be following a similar work submission and review process to previous module exercises.

Please make a folder called "documentation" in the root of your project directory, place your completed diagrams in this folder, naming them appropriately, and commit them.

We'll be using Pull Requests on GitHub in the same fashion as the last module to review this exercise submission.

If you experience issues, ask a tutor for help!

Part 3: Component Diagram

Add a Component diagram. The component diagram should zoom in on a specific component of the container diagram. In this case, it should focus on your own application code.

Component diagrams are detailed. In a well-structured source code repository, the blocks of your component diagram may reflect different Python sub-packages, and illustrate how they are used.

Beware: When looking at examples online you may see the term "single page application" and mistakenly think that's what you have. "Single page application" refers to a web app architecture where the server (e.g. your Flask app) serves a page with JavaScript. The JavaScript runs in the user's browser and sends requests to the server when you submit forms for example, without reloading the page. It even makes it look like the user is navigating between pages while never actually fetching new HTML from the server, hence the term "single page". In contrast, assuming you have just been following to the exercise instructions, you have a "multi-page app" where the server does everything and serves templated HTML. It just happens to have only one page for simplicity's sake.

(Stretch Goal): Code Diagram

The final layer of the C4 model is the code diagram. This is very low-level, and documents the interactions between different functions and classes. You shouldn't try to draw this yourself!

IDEs and code editors can generate code diagrams for existing systems for you. Because they are highly-technical and deterministic, code diagrams usually use a more strict modelling language than our "arrows and boxes" sketches. Unified Modelling Language (UML) is popular for these types of diagramming tasks.

For this stretch goal, explore ways to generate code diagrams from VSCode. Can you create one for your todo app?