**Slide 1 – Title**

Hey guys, we’re Ailurus and our product is… Ailurus!

**Slide 2 – About Ailurus**

So, about Ailurus:

For individuals or small businesses, setting up a presence on the web can be a daunting, risky task when you have a light budget and little technical knowledge.

That’s why Ailurus aims to be a versatile solution covering the software, hardware, and service aspects.

First, we provide a web platform. This includes an administration interface for managing your web server and an app store for installing web applications.

Secondly, we provide a personal home server. This is in the form of a lightweight device, such as the raspberry pi, which you can run at home to host your server. This will be simple to set up and will give you 100% control of your data.

Thirdly, we plan to offer a service to provide domain names. We’ll let the user select what they want their server to be called, and we’ll hook up everything for them.

**Slide 3 – Related Products**

While performing market research, we came across several related products.

These come from well-known competitors such as Synology, QNAP, and Windows Home Server.

Synology and QNAP are similar in that they both specialize in Network Attached Storage devices. Their main focus is on a high quality NAS, but they also provide an administrative web platform with their own app stores, making their products hardware/software solutions.

The Windows Home Server is a pure-software solution, but it has been discontinued and is no longer supported.

**Slide 4 – Our Focus**

We intend to differentiate from these products by targeting a specific audience that they do not reach.

Our focus is on being affordable, simple, and extensible.

Firstly, we’re aiming for a much lower price point. We’re looking at about 50$, which is very inexpensive and low-risk, compared to Synology and QNAP. A NAS from one of these companies could be one-hundred to two-hundred, or even several hundred dollars.

Secondly, we’re aiming for simplicity. Not only is all of the setup and configuration handled behind the scenes, but we’re stressing a sleek, clean user interface that’s easy to understand and thus easy to use. We help the user get online, provide ready-to-install web applications, and the user won’t have to worry, or even know, about all of the dependency management we’re taking care of. It’s a super low technical barrier-to-entry.

Finally, in addition to the apps we provide to the user, such as Wordpress or ownCloud, we provide an API and build tools so that third-party developers can easily bundle new apps. In this way, users have access to an unlimited number of apps.

**Slide 5 – Repo Layout**

Our repo is hosted on git hub and facilitated through Arcanist and Phabricator

The Git workflow is as follows

* Each feature is developed separately on a new local branch
* The branch diff is sent to Phabricator for code review
* Once approved, system will merge branch into master

**Slide 6 – User Interface**

DEMO:

🡪 URL will take us to Phriction, where we have a compilation of our current mock ups for the finished product

🡪 We should show off Slowvotes so they can see how our deliberation process works

**Slide 7 – Technologies**

Here is a list of the technologies we’re working with:

We’re using Phabricator to handle our project organization, including code reviews, voting on design alternatives, and work history.

For the web server, we’re using Maven for dependency management and project building. Our web interface itself is constructed using Google Web Toolkit. And our actual web server is being hosted using Jetty.

Our back end is running with Python3 setuptools. Testing is done with Pyunit. And the hardware is set up using a custom bootstrap script that takes a fresh Debian install and configures it into a workable state.

**Slide 8 – Hardware Details**

For hardware, we started with a Raspberry Pi Model B and picked up a Cubieboard 2 as an alternative.

We’re currently developing on both to consider the differences and how the difference hardware specifications affect the course of our development.

Overall, the Cubieboard’s processor is slightly more powerful, but we can match a similar strength on the Raspberry Pi by overclocking it.

The Raspberry Pi comes with about half as much memory as the Cubieboard.

However, while the Cubieboard’s specifications are generally higher, it comes at a literal cost of… Cost.

Averaging about 60$, we instantly overshoot our targeted price point of roughly 50$, so we’d like to try further optimizations on the Raspberry Pi before switching to this alternative.

**Slide 9 – Software Details**

So, for our front end, we’re writing the web interface in Java with Google Web Toolkit.

This gives us access to a robust set of widgets and allows us to compile Java code into highly optimized Javascript.

All of the dependency management for our code is managed by Maven, which also performs our builds. Maven allows us to package our project into a .war file which may then be deployed by Jetty.

On our hardware, we’re running Debian OS. At the moment, we’re using a regular system, but in the future we’ll be looking to optimize it to make it lighter.

Our packaging system runs on Python3, which lets us easily create build tools and expose a simple API for package management.

Finally, our web server is using nginx for serving pages by delegating requests to Jetty, and Jetty, once again, handles the deployment of our project which is written in Java.

**Slide 10 – Packaging System**

For our packaging system:

* First, applications are wrapped as bundles containing an installation script and an configuration files needed for setup, as well as some basic information such as a download source and validation checksum
* Now, we expose a setup tool that may be invoked via command line that will install a specified app
  + This is done using our custom command “ai-get install path/to/package”
* During the installation process, the packaging system will check an application cache to see if the app has been downloaded previously.
  + If the application isn’t found, or we find that it is corrupted due to a mismatching validation checksum, we download the application from a URL provided by the application bundle
* Finally, after the application is verified, we copy over the necessary files and complete any configuration that needs to be done

**Slide 11 – Iteration Plan**

UI Mockups – Since the user interface is such a strong focus for us, we went through several different design mockups. We eventually decided on a style similar to what you’re seeing right now.

Two Hardware Prototypes – We initially bought a Raspberry Pi to work with, but we began development on a Cubieboard as a possible alternative.

Working Packaging System – We wanted a simple, yet robust, way to install applications so that it could all be handled without user configuration.

Research Competition – We had to scope out related products to see our competition and decide what our main focuses would be in order to differentiate.

Basic Admin Web Server – Essentially the crux of our product, we needed to get an administrative web server working for us to build off of.

There are a few objectives that were uncompleted and fall into our backlog.

Consult Customers – We spoke with a few interested customers last year, but have yet to reconvene with them in order to evaluate our direction.

Domain Name Service – This service will likely be realized later in the project lifecycle.