# **1 Slide – Second title slide**

Hi, I’m Mandy Hubbard.

I’m a QA Architect at Care.com Homepay

and I’m really excited to be here today to share with you how our engineering team is using Jenkins to build containerized microservices. How many of you are using Jenkins for your continuous integration and delivery pipeline?

These pipelines allow us to build and release well-tested software reliably and quickly. However, if our continuous integration and delivery platform became unavailable, we would be unable to release our software at all.

Our team uses Jenkins to build and deploy software every day, and Jenkins is critical to our organization’s success.

For our team, Jenkins is indispensable.

Scenario

Let me paint a picture for you. - It’s 4:30pm on Friday, you have a major release on Monday, and your Jenkins server goes down.

It doesn’t matter if it was a hardware failure, a fat-finger accident, or the zombie apocalypse - your Jenkins server is toast. How long did it take to perfect your Pipeline, all your Continuous Delivery jobs, plugins, and credentials?

Hopefully you at least have a recent backup of your Jenkins home directory, but you’re still going to have to work over the weekend with IT to procure a new server, install it, and do full regression testing to be up and running by Monday morning. Let me see a show of hands - How many of you have experienced this? Or maybe it’s your worst nightmare.

Well guess what? For the engineering team at Care.com HomePay, Jenkins is indispensable and completely disposable. It has to be. It’s too important to our organization.

I know….indispensable and disposable sounds contradictory. But I’m going to show that it’s not…and that it’s the best way to maximize Jenkins in your organization.

# **2 Slide - Care.com Homepay**

But first, let me share with you why it’s important to Care.com HomePay.

Care.com is the

world’s largest online platform for finding and managing family care.

Many families use our services to find nannies, babysitters, pet care and elder care.

We also help companies provide care benefits to their employees.

The HomePay division is the largest provider of payroll and tax services for household employees – caregivers -- in all 50 states. Since we move money between accounts, we are subject to many of the restrictions and regulations of a financial services company. Our processes must be solid.

And we’ve been doing it for 25 years.

HomePay has innovated in creating this industry and now we’re innovating in the way we do our jobs. Caregivers depend on our services to get their paychecks and our clients depend on us to deliver these paychecks and to stay on top of all tax filings on time. That’s our pledge to them and we need to make sure we can reliably release the software that provides this service. There’s a lot at stake here.

Last summer I joined a new team at Care.com HomePay that was tasked with two things:

1. Deconstruct a monolithic application into containerized microservices.
2. Transform IT from a traditional, waterfall-based cost center into an agile and dynamic asset that could help the business achieve its vision of digital transformation.

The first task required to achieve these goals, especially with containers and microservice, was to develop a reliable continuous integration and delivery pipeline. And for this pipeline, I chose Jenkins.

# **3 Slide – Bio**

And this leads to why this is important to me.

I’ve spent over 15 years of my career leading software QA teams and being held accountable for the quality of the software that was released. I also spent a year and a half as a Support Engineer, which confirmed that I would much rather work to ensure quality before the software is released than deal with poor quality after it’s in the hands of the customer. I’m in this because I care about quality.

# **5 Slide – What is quality?**

Everyone says they care about quality, but what does that really mean? More specifically, what does it mean for us in this room who are responsible for ensuring that the code that we build runs in the way it was intended?

For me, quality is the ability to release software in a reliable manner that reduces the risk of regression.

I want us to think about quality when defining our processes, not just in testing the end product,

not waiting until the end of a release cycle to find out that we have a million merge conflicts to resolve.

That’s continuous integration. The next level is continuous delivery, which I’ll show you soon in a demo.

Every business cares about quality. Why? Because quality is what allows us to move fast.

## **6 Slide - Why do cars have brakes?**

How many of you have heard the old adage – why do cars have brakes? So that they can go fast.

Without brakes, the only way to stop the car would be to coast to a stop, meaning that the car could never go full throttle and still be able to stop reliably.

Likewise, it’s building quality into the software and following processes with built in quality checks that allows developers to confidently deliver software quickly. The business is able to move at lightening speeds, right up against the edge of safety.

Fortunately, releasing quality software is simple, right?

# **7 Slide – Steps to quality software**

No, it really is. Developers commit code regularly, preferably one or more times a day to a shared mainline branch. Each commit triggers an automated build and test and is only merged to the shared mainline if all quality checks pass.

You need a well-defined process and developer discipline. The problem with this is that developers are human. We make mistakes. We also may feel tempted to cut corners in our well-defined process when struggling to meet deadlines.

The good news is that with Jenkins we can automate not just the building of our software but the entire end-to-end release process so that minimal human intervention is required.

Developers love that they don’t have to worry about how the software gets released so they can focus their energy on writing code.

# **8 Slide – Github Process**

Let me describe what this looks like on our engineering team. Our team uses Github as our source control management system.

We create individual repos for each of our projects under our organization’s Github account. This is my upstream repository. When I’m ready to develop a new feature for a particular project, I create a fork of the project’s repo under my own Github account. I do all my development in a branch on my fork and when my code is complete, I open a Pull Request for my code to be merged into my upstream master.

# **9 Slide – Continuous Integration**

With this configuration in place, Github sends a notification to our Jenkins server via the webhook that the Pull Request has been opened. Jenkins then builds the pull requests and runs automated tests. While Jenkins is building the pull request, it cannot be merged into the mainline. The “squash and merge” box is replaced by another box indicating that not all quality checks have passed. If those quality checks fail for any reason, whether the code fails to build or there is a failing test or maybe the code coverage target is no longer being met, the box turns red and the code cannot be merged to master. If the build is successful, Jenkins notifies Github and the “squash and merge” box returns. A slack message is sent to our dev team notifying them that there is a PR waiting for code review.

# **10 Slide - Continuous Delivery**

Once the PR is merged into master, Github sends another notification to Jenkins. At this point, Jenkins builds the master branch, running all the same quality checks. It can also run an additional battery of tests since this is our release branch – maybe a set of regression or performance tests that are run on release candidates. Once that build is complete, Jenkins tags the commit in Github, pushes the versioned Docker image to our Docker registry and deploys the new image to production. That’s it.

## **11, 12 Slide -Github Webhook**

Each of our repositories is configured with a webhook that allows Github to communicate with our Jenkins server. We specify a publicly accessible URL for our Jenkins server and select the events we would like to be notified of. In our case, we want to be notified of pull requests or pushes to the master branch.

# **13 Slide – Protected Branch**

We release from our master branch so we have locked down this branch so that code cannot be merged into it unless it has passed our defined quality checks – in this case our Jenkins PR merge.

# **15 Slide - Demo**

What does this look like? How does my team use Jenkins every day? Let’s look at how Jenkins is indispensable to our continuous deliver process.

# **16 Slide -Demo**

LDAP

Global Pipeline

Slack

Docker

Github Organization Plugin

Remember the thing that makes Jenkins so flexible is its plugin environment. If it tried to provide everything anyone could ever need right out of the box, it would be both too much and not enough all at once. I’m about to show you MY favorite plugin – the Github Organization plugin.

Pull request kicking off the build.

Merge to master.

Jenkins building master branch.

Return to slides.

Do you see what we’ve done here? We’ve built an indispensable, disposable Jenkins, just like I promised.

The best part is that developers like using it. Everyone agrees on it because it is easy to use, gets rid of a lot of manual steps and is reliable. We aren’t forcing tools and processes on people, we are changing the culture, which is at the very heart of DevOps.

My team believes in our continuous delivery process, so it’s easy to get them to follow it. Jenkins is doing this. Jenkins is critical to our way of life. It’s so critical that we can’t afford to lose it even for a few hours, not to mention the time it would take to rebuild this monster. It’s indispensable.

We’ve automated the installation and configuration of plugins and the creation of credentials sets and jobs. Everything required to set up a new Jenkins environment is stored in source control, versioned, and released just like any other software.

Now let me show you how we’ve made Jenkins disposable.

# **17 Slide – Technologies**

In addition to Jenkins, of course, the technology stack we used to create a disposable Jenkins includes Docker, ContainerPilot, Consul and Docker Compose.

Let’s first talk at a high level about the technology we used and make sure everyone has some context.

Docker – Who here is using Docker?

As many of you know, Docker is a container platform. Running Jenkins in a container allows to automate a lot of repetitive tasks and package Jenkins in a portable way.

We happen to be using Joyent as our cloud provider but you can run this anywhere – AWS, Azure, Google, anywhere.

ContainerPilot

We’ll be talking a lot about ContainerPilot. Just a show of hands. How many of you are familiar with the AutoPilot pattern?

Containerized applications that implement the autopilot pattern are responsible for understanding their own startup, shutdown, scaling, and recovery from failure and are smart enough to reliably and accurately configure themselves for operation with no human intervention.

They are completely automated. ContainerPilot is Joyent’s implementation of the AutoPilot pattern. It’s a helper application that runs inside your Docker container to facilitate the autopilot behavior.

ContainerPilot acts as a supervisor process that monitors the application, registers it with a service catalog, in our case Consul, performs health checks and runs user-specified code at events in the life cycle of the container.

For example, we can specify an action to be performed before the application starts, and before and after it stops. This life cycle management feature is the one that allows us to preconfigure our Jenkins container before the application is started by specifying a script to run at the preStart event.

The reason that’s important is because there is a lot of automated configuration that has to happen each time we bring up a new container and tying them to the preStart lifecycle event in ContainerPilot is what ensures that the process is reliable and repeatable.

Consul

Consul is a service discovery application. For the demo, I scaled down the application cluster to just Jenkins and Consul, but in our production Jenkins environment, we use Consul to discover other services we rely on such as Vault for managing secrets and Traefik for reverse proxy and load balancing.

Docker Compose – How many of you have used Docker Compose?

Docker Compose allows us to define and run multi-container Docker applications and configure the services uses a compose file.

# **18 Slide -Architecture**

When the container starts, both ContainerPilot and the Consul agent are started and we are in the preStart event of the lifecycle of the container. We’ve specified a script to run during the preStart event and when it completes, ContainerPilot starts the Jenkins server.

We can also specify health checks to run at pre-defined intervals. This would enable us to detect a situation where the container was still running but the Jenkins server was not functioning properly. And finally we can define other tasks to run throughout the life of the container.

# **20 Slide – Docker Compose**

Let’s talk a bit more about Docker Compose. Docker Compose allows us to configure containerized services using a compose file.

It also allows us to inject data into our running containers via environment variables passed in through an environment file.

This is a snippet from my docker-compose.yml. In it I specify the image to run, and the ports to map. I use it to mount the docker socket from my local Docker host so that Jenkins can use it to build additional Docker images. This is an alternative to running Docker in Docker if any of you are familiar with that.

Finally, the most important pieces for creating a disposable Jenkins environment are the specification of the \_env file, which is how I will pass information to my running Jenkins container and the command.

Notice that I’m not running the Jenkins server directly. Rather I am running the ContainerPilot script and passing ‘jenkins.sh’ as an argument. Remember how I said that ContainerPilot acts as a supervisor process? ContainerPilot starts when the Jenkins container is started and then starts Jenkins only after it has performed preStart configuration and registered it with Consul.

Transition

Now, let’s dive into the details. I’m going to show you how I automated the configuration of the LDAP plugin as an example. I’ve used this same approach to automate all the other plugins we saw in the demo and you can you this approach to automate your own set of plugins.

# **21 Slide – Anatomy of a Plugin**

Plugins are typically configured by entering data in one of the screens in the Jenkins UI. The configuration data entered through the UI is stored in an XML file on the file system. The plugin then reads information from the XML file when it runs.

## **22 Slide – LDAP Plugin UI**

I’m going to use the LDAP plugin as an example in my explanation, but you can use this same approach with any plugin. Recall from the demo, the screen used to configure the LDAP plugin.

## **23 Slide – Configuration File**

Which results in this XML file. Different plugins write to various xml files but they all operate this way.

### Zoom

We need to specify values to customize this plugin for our environment. Where do we get this information?

## **24 Slide – Docker Compose Environment File**

We pass them into the Jenkins container at run-time using a Docker Compose environment file.

So let’s recap: we configure plugins by modifying values in an XML file. We have all the values we need in environment variables.

## **25 Slide – firstRun.sh**

We then run a script that adds our data to the XML file for the plugin. This is a snippet from the bash script we run in the preStart event. We use a tool called xmlstarlet to update the value of each of the relevant paths to the values stored in our environment variables. Jenkins uses its own encryption algorithm specific to this running instance to encrypt the manager password secret.

# **26 Slide – ContainerPilot Config**

Finally, we specify in the ContainerPilot configuration file that this script be run in the preStart event. This script contains methods for making the appropriate data substitutions for all the plugins we want to preconfigure.

# **27 Slide – Summary**

Why did we do this? What’s the point? What does the business want?

We did this because the business wants to go faster. They are always asking us to go faster, faster, faster. With Agile and DevOps we can make the business go faster but quality and continuous delivery help insure that we’re doing that safely.

Jenkins is how we achieve quality.

In order to create our completely automated delivery pipeline, our team has utilized Jenkins advanced capabilities to the fullest, but Jenkins is software. Roll up your sleeves and get in there and you’ll see that it’s even more versatile than you knew.

Our Jenkins server is critical. It took us weeks to perfect it and we can’t afford to repeat this is something goes wrong.

Your Jenkins server is critical, too. Even if you don’t yet have a full continuous delivery pipeline built out, the functions your Jenkins server are providing right now are critical to your team.

The more automation we add and the more dependent we become on Jenkins, the greater the risk of allowing Jenkins to become the single point of failure. The only way to remove that risk is to make the thing itself disposable.

Remember the nightmare scenario we started with at the beginning of the session? The Jenkins server crashes? If that happens to Care.com HomePay, we're good. Our world keeps moving and we sleep just fine at night. By building a Jenkins environment that’s both indispensable and completely disposable, you can too.

"Thank you for attending my session, if you can please log onto the mobile app which is called Jenkins World 2017 and leave some feedback for me I'd love to hear what you thought. You can do this by touching on "Post-session survey" under the session description"

# Pre-Demo Checklist

1. Start LDAP

sudo kill $(sudo lsof -t -i:389)

docker start 7e

1. Start Ngrok

From /home/mandy/Documents

./ngrok http -subdomain=jenkinsworld 8080

1. Github

Have a PR ready

Make sure there is only one branch and only one PR

1. Check system clock

sudo ntpdate ntp.ubuntu.com