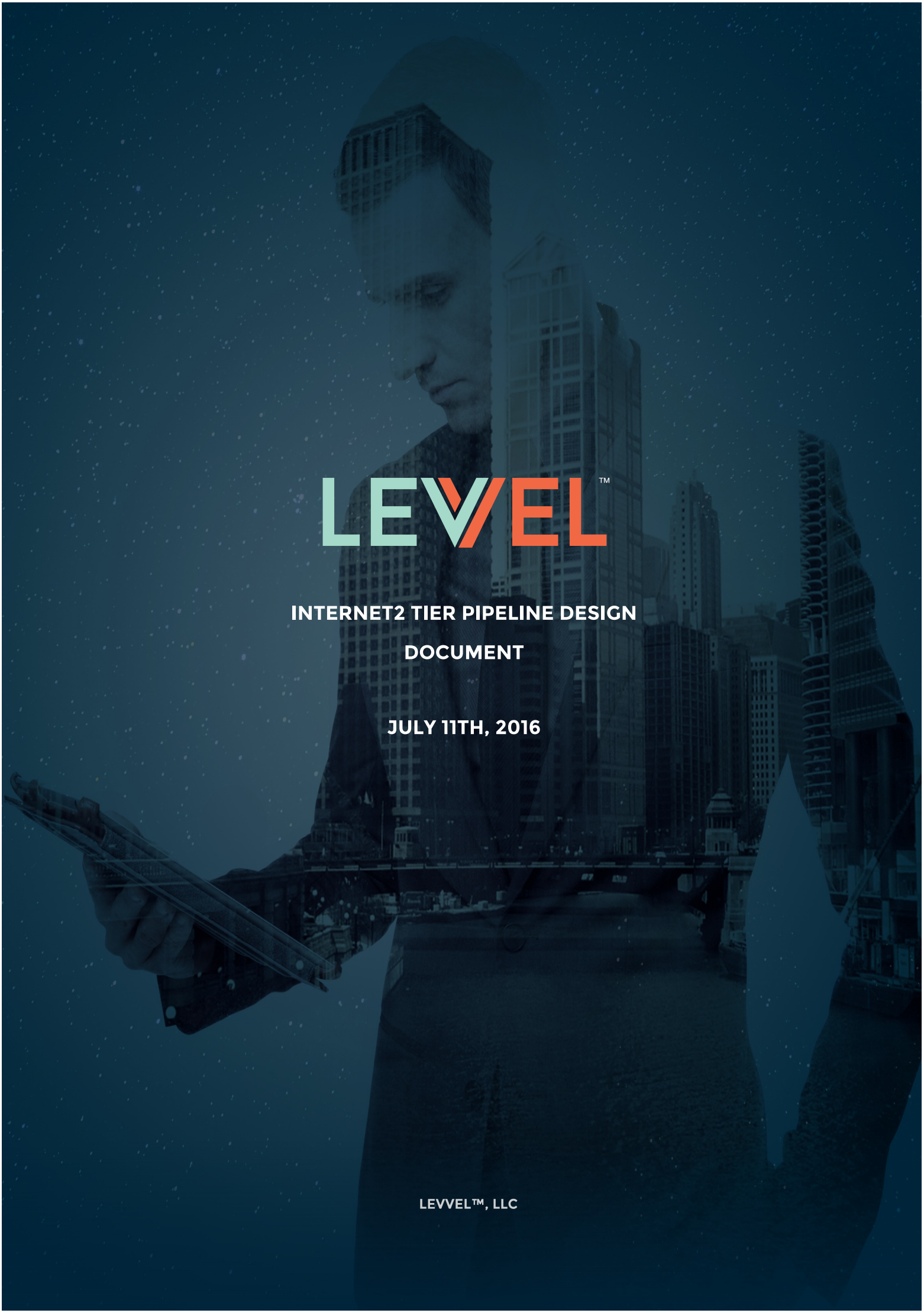

LEVVEL™

INTERNET2 TIER PIPELINE DESIGN DOCUMENT

JULY 11TH, 2016

LEVVEL™, LLC

BUILD/TEST PIPELINE DESIGN DOCUMENT

Prepared by: Jim Van Fleet (jim.van.fleet@levvel.io)

Approved by:

OVERVIEW

The TIER group of the Internet2 consortium has contracted with Levvel to produce an appliance that packages the Shibboleth IDP software in a way that maximizes the ease with which member institutions can begin using that software in production environments. The appliance in question is to be delivered via a build pipeline that automates the construction of the appliance from its various components. Additionally, the appliance is to use Docker to run any and all service processes required for proper software function.

This document is intended to describe the mechanisms via which container images are constructed and tested for viability, and how, in turn, the appliance are constructed from those .

- Container Construction
- Build Pipeline
- Testing Approach
- Publishing a Container Image
- Appliance Construction
- Distributing the Appliance
- Still to Resolve

CONTAINER CONSTRUCTION

While the details of container construction are best left to their own documents, we'll note some of the important elements of those containers regarding the rest of the pipeline.

LOCATION

We've created a 'docker' organization on Github Enterprise, and public repositories for several of the desired container images therein. Each container repository will contain a Dockerfile, files and folders that support construction of the container image by the Dockerfile, and a Jenkinsfile that defines the build pipeline process.

BUILD PIPELINE

Though Docker is a current focus of the TIER group, the group's long-term commitment is for an appliance (or appliance-like) mechanism that makes onboarding with Shibboleth IDP easier. Regardless of the technologies surrounding the appliance, a build pipeline that automates the process of testing and constructing the appliances should outlive any particular approach to its contents.

HOSTING ENVIRONMENT AND PROVISIONING

The build pipeline will operate within TIER group's AWS environment, and the instance running Jenkins will be automated to a significant degree in the 'docker/ansible-playbooks' repository on I2's GitHub Enterprise installation.

COMPOSITION AND CONFIGURATION

Jenkins is a well-regarded, open source continuous integration server already adopted by other working groups at Internet2. In its latest release, Jenkins offers a Pipeline feature that allows the behavior of the build pipeline to be defined in source code rather than by Jenkins administrators. In addition to empowering developers to define and manage stages of the pipeline over time, it unlocks the powerful collaboration features of Jenkins and GitHub to facilitate development of the pipeline itself.

Additionally, Jenkins and GitHub Enterprise are commonly used together and there are several helpful points of integration between the two. These include (but are not limited to) the following:

GitHub Enterprise as OAuth provider

This permanently delegates user management within Jenkins to the GitHub Enterprise installation. Permissions may still be managed within Jenkins, but uses reasonable defaults through a "GitHub committer strategy" that selects reasonable access levels based on the user's access to the GitHub repository.

Source and triggers

During our primary development, triggers will resolve automatically as development proceeds. Development builds will not trigger the construction of an appliance, and may not publish a container image to Dockerhub. In the event a container image is published to Dockerhub, it will be clearly labelled and tagged so that it cannot be used by the general populace.

At this time, the appliance build will be triggered manually. It will track official Shibboleth IDP releases.

TESTING APPROACH

Our build pipeline is designed both to facilitate the development of container images in the short term and to construct appliances in the long term. During development, responsive triggers and unit testing can identify and help eliminate most errors quickly. While many of our 'units' are straightforward, they may be as complex as an interdependent service with a large configuration surface area. For more complex cases, we will supplement our unit testing with integration testing.

UNIT TESTING

In the repositories for each container image, our pipeline engineer will be able to create scripts that provide a variety of assurances regarding proper function. Examples include, but are not limited to, the following:

- Ensuring that running a container image leaves a running docker process
- Ensuring the running container binds to a defined port on the host
- Ensuring that a command executed inside the container exits successfully, or produces desired output.

We further anticipate that we will be able to collate these results and display them in aggregate over time to help determine if we need to make certain aspects of the container image easier to develop with.

INTEGRATION TESTING

If a container image passes its unit tests, it then becomes appropriate to test that container image in a controlled environment simulating real-world conditions. The process of creating an appliance generates a meaningful intermediate output that allows us to perform integration testing on EC2 with relative ease.

Since this is time- and cost-intensive, we plan to build this capability, but not to perform it as a part of every check-in. We recommend decisions regarding timing of integration testing be evaluated on an image by image basis.

PUBLISHING A CONTAINER IMAGE

If a container image has passed its battery of tests, we recommend distributing the image via Dockerhub (or other Docker registry). This process can be performed automatically by the pipeline with contextually appropriate additional tags or labels depending on the context of the triggered build.

APPLIANCE CONSTRUCTION

After a container image has been published, we can leverage that container image from inside an appliance we construct. Our target appliance is the VirtualBox OVF (or Open Virtualization Format) which is widely compatible with VMware as well. Packer from Hashicorp offers native support for constructing appliances of this kind.

Because our build pipeline runs on EC2 (itself a virtualized environment), however, direct access to binaries needed to support packer in using VMware or VirtualBox builders cannot be used[1]. Our approach uses additional EC2 and AWS offerings to produce the desired outputs. The phases are as follows:

CREATING AN OVF TO IMPORT TO AWS

This phase is conducted on a developer laptop in our offices. We configure packer to use the VirtualBox ISO builder, a distribution ISO from CentOS, and a kickstart file to allow packer to continue through installation without a graphical interface. The output of the packer process is an OVF image that we can import into AWS, where it's exposed for use as an AMI

USING THE AWS-EBS BUILDER

The Packer aws-ebs builder uses the AMI output by phase 1 as a starting place, then uses ansible as a provisioner to install and configure all required software for the appliance. Finally, the instance is stopped

and the volume snapshot is used to construct another AMI.

EXPORTING AN OVF FROM AWS

In the third phase, using the AWS EC2 export functionality, we create an OVF from the provisioned AMI. This functionality is only available to AMIs that were originally imported, which is why we must create an OVF in phase 1. In addition to the desired OVF output in an S3 bucket we can configure to be publicly available, this process realizes an additional benefit of creating an appliance AMI that can be used both in our integration testing and also by the larger community.

DISTRIBUTING THE APPLIANCE

We have so far deferred discussion on delivering successfully built appliances.

STILL TO RESOLVE:

---

[1]We have not verified that VMWare Workstation Pro is incompatible with AWS, but it is inconvenient enough that we are evaluating the approach described herein first, as it has other benefits. VirtualBox certainly cannot be used on EC2.