

OPENNTF WEBINARS

May, 2022 OpenNTF Webinar

Using The New Domino One Touch Setup



AGENDA

- Welcome – Howard Greenberg and Graham Acres
- Presentation Project – Oliver Busse
- Presentation – Roberto Boccadoro and Jesse Gallagher
- Q and A - All



THANKS TO THE OPENNTF SPONSORS

- HCL made a contribution to help our organization
 - Funds these webinars!
 - Contests like Hackathons
 - Running the organization
- Prominic donates all IT related services
 - Cloud Hosting for OpenNTF
 - Infrastructure management for HCL Domino and Atlassian Servers
 - System Administration for day-to-day operation



THIS IS OUR COMMUNITY

- Join us and get involved!
- We are all volunteers
- No effort is too small
- If your idea is bigger than you can do on your own, we can connect you to a team to work on it
- Test or help or modify an existing project
- Write guides or documentation
- Add reviews on projects / stars on Snippets



OPENNTF BOARD UPDATES

- Community Projects
 - Catalog of User Group Presentations
 - Led by Oliver Busse
 - Channel on slack.openntf.com #presentation-project
- The Future of OpenNTF
 - How to Evolve OpenNTF
 - We want your input!
 - Blog and video posted soon
 - Feedback via Discord

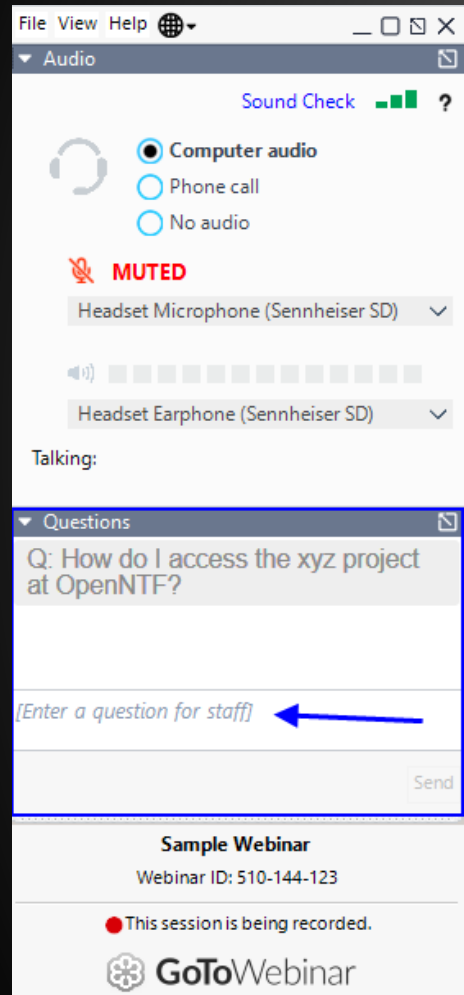


NEXT WEBINAR

- Watch <https://www.openntf.org/webinars> for more information
- June 16th: OpenNTF Annual General Meeting



ASKING QUESTIONS



- First Question – Will this be recorded?
 - Yes, view on YouTube!!!
 - <https://www.youtube.com/user/OpenNTF>
- Use the Questions Pane in GoToWebinar
- We will get to your questions at the end of the webinar
- The speakers will respond to your questions verbally
 - (not in the Questions pane)
- Please keep all questions related to the topics that our speakers are discussing!!!
- Unrelated Question => post at:
 - <http://openntf.slack.com/>



ONE TOUCH SETUP

Roberto Boccadoro and Jesse Gallagher



ONE TOUCH SETUP FOR DOMINO V12

Roberto Boccadoro – OpenNTF Contributing Director
ELD Engineering



WHAT IS ONE-TOUCH SETUP ?

In previous versions of HCL Domino, setting up a Domino server involved multiple steps. Starting with Domino 12, you can use one-touch Domino setup to set up a server in a single step.

You invoke one-touch Domino setup by referring to a JSON file or a set of environment variables that contain the setup configuration information.

Using one-touch Domino setup you can:

- Set up servers
- Set up an ID vault
- Create and update applications and documents and enable and run agents. This feature is available only through JSON file input.

One-touch Domino setup is supported on Domino on Docker, Windows, and UNIX platforms.



DOCUMENTATION AND SOME EXAMPLES

https://help.hcltechsw.com/domino/12.0.0/admin/inst_onetouch.html

https://help.hcltechsw.com/domino/12.0.0/admin/inst_onetouch_example_servervaultapp.html

<https://github.com/nashcom/domino-startscript/tree/main/OneTouchSetup>



DEMO TIME!



DOMINO ONE-TOUCH SETUP FOR DEVS

WHY DOES IT MATTER FOR DEVS?

- **Same reasons it matters for admins - most of us dip into Domino admin eventually**
 - **Easily create dev servers**
 - **Consistency with production**
 - **Combine with Docker for an even-better version of those!**
 - **Combine *that* with automated tests**
-

REFERENCES

- https://help.hcltechsw.com/domino/12.0.0/admin/inst_onetouch.html
 - <https://frostillic.us/blog/posts/2022/1/23/building-a-full-domino-image-for-junit-tests>
 - <https://github.com/OpenNTF/org.openntf.xsp.jakartaee/tree/2.4.0/eclipse/tests/it-xsp-jakartaee>
-

DEV SERVERS

DEV SERVERS

- **Domino isn't difficult to install, but this smooths the process all the more**
 - **Since configuration is in a JSON file, the more you put in there, the more will be consistent between multiple installations**
 - **Configuration Documents can do *much* of this, but not all of it**
 - **Make sure to put the JSON file in Git!**
 - **(Tell your admins to do that too)**
 - **Use this to create commonly-used dev databases, like the Log Reader from OpenNTF**
-

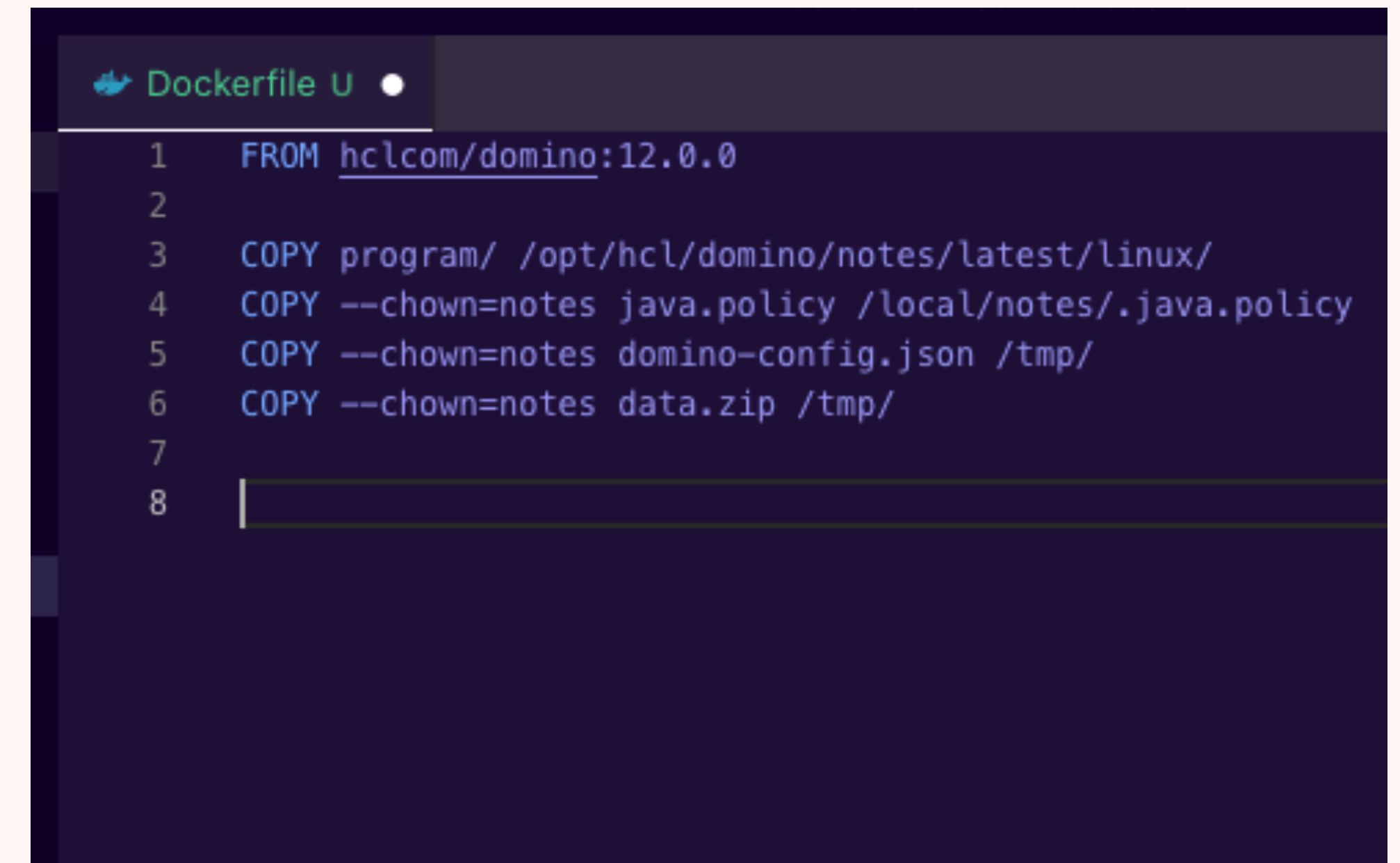
CONSISTENCY WITH PRODUCTION

- **Production may differ from a normal dev server in common ways:**
 - **Use of DAOS, NIFNSF, etc.**
 - **Server "Security" tab settings**
 - **Standard deployed databases**
 - **Derive your configuration JSON from the production one to keep things common**
 - **This gets all the easier with Docker deployments**
-

DOCKER

DOCKER

- **Natural fit for both production and development**
- **Both the official Flexnet images and the "community" script from GitHub will work**
 - **The GitHub script also lets you deploy to the data dir, which is handy**
- **This helps cover configuration outside of the One-Touch Setup file too:**
 - **Java Policy**
 - **Custom JARs**
 - **Custom Tasks/ExtMgr Addins**



```
Dockerfile U
1 FROM hclcom/domino:12.0.0
2
3 COPY program/ /opt/hcl/domino/notes/latest/linux/
4 COPY --chown=notes java.policy /local/notes/.java.policy
5 COPY --chown=notes domino-config.json /tmp/
6 COPY --chown=notes data.zip /tmp/
7
8
```


DOCKER COMPOSE

- **Makes it easier to manage related volumes - DAOS, etc.**
- **Though Domino deployments are usually single-server, the format of the config file isn't overkill**
- **But this would sure make setting up associated servers easy too**

```
docker-compose.yml U •
1  services:
2    devserver:
3      build: .
4      ports:
5        - "1352:1352"
6        - "80:80"
7        - "443:443"
8      volumes:
9        - data:/local/notesdata
10       - ft:/local/ft
11       - nif:/local/nif
12       - translog:/local/translog
13       - daos:/local/daos
14      restart: always
15      environment:
16        - LANG=en_US.UTF-8
17        - CustomNotesdataZip=/tmp/data.zip
18        - SetupAutoConfigure=1
19        - SetupAutoConfigureParams=/tmp/domino-config.json
20  volumes:
21    data: {}
22    ft: {}
23    nif: {}
24    translog: {}
25    daos: {}
```

TEST SUITES

TESTCONTAINERS

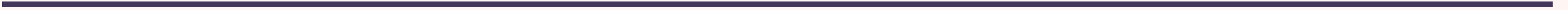
- <https://www.testcontainers.org/>
 - **Open-source library for deploying temporary Docker containers during JUnit test suites**
 - **Most commonly used for dependency servers, like PostgreSQL, or "true" Selenium browsers**
 - **Can get a little fiddly to set up for Domino, but it works**
-

TESTCONTAINERS

- **Pairs well with:**
 - **Mavenized OSGi plugins - deploy them during Docker build**
 - **NSF ODP - build with Maven and deploy+sign in the One-Touch JSON**
 - **Selenium - use true Chrome and Firefox engines to test web apps without the setup hassle**
 - **Fits most naturally with HTTP-based tests, but could use other protocols**
 - **LDAP, IMAP, etc.**
 - **NRPC client -> server or server -> server**
 - **Custom ports if you're developing an addin that exposes one**
-

TESTCONTAINERS SETUP

- **Can use a Dockerfile, Java-based configuration, or both**
- **Testcontainers allows you to reference project resources to add to the container**
 - **Good for programmatic needs, like deriving the Maven version to find the built update site**
- **Can also use filesystem binds, which can be handy to keep image stages reusable as long as you're working locally**



OSGI DEPLOYMENT

- **OSGi deployment can be tricky: the official image doesn't have a way to deploy random files to data, while the community image makes timing fiddly**
- **Equinox deployment links to the rescue!**
 - **Copy/bind the update site to scratch space in the container, e.g. /local/eclipse**
 - **Create a ".link" file containing "path=/local/eclipse"**
 - **Copy/bind to /opt/hcl/domino/notes/latest/linux/osgi/rcp/eclipse/links**
 - **They'll be picked up on first HTTP start**

<https://github.com/OpenNTF/org.openntf.xsp.jakartaee/blob/6cc36ef5b5376a8185dcec03aa57a0525ef9cace/eclipse/tests/it-xsp-jakartaee/src/test/java/it/org/openntf/xsp/jakartaee/nsf/docker/DominoContainer.java#L100>

CONTAINER INIT

- Testcontainers uses "WaitStrategy" rules to tell when a server is up
- A Domino deployment may need a combined rule:
 - Wait for "Adding sign bit" from AdminP signing
 - Wait for "HTTP Server: Started"
 - Wait for any custom deployment agents/plugins
- With the community image, set **DOMINO_DOCKER_STDOUT=yes** env variable

```
waitingFor(  
    new WaitAllStrategy()  
        .withStrategy(new LogMessageWaitStrategy()  
            .withRegex(".*Adding sign bit to.*") //$  
            .withTimes(300)  
        )  
        .withStrategy(new LogMessageWaitStrategy()  
            .withRegex(".*HTTP Server: Started.*") /  
        )  
        .withStrategy(new LogMessageWaitStrategy()  
            .withRegex(".*Done with postinstall.*")  
        )  
    .withStartupTimeout(Duration.ofMinutes(3))  
);
```

<https://github.com/OpenNTF/org.openntf.xsp.jakartaee/blob/6cc36ef5b5376a8185dcec03aa57a0525ef9cace/eclipse/tests/it-xsp-jakartaee/src/test/java/it/org/openntf/xsp/jakartaee/nsf/docker/DominoContainer.java#L61>

ACCESSING URLS

- **Launching the container will provide a "GenericContainer" (or subclass) instance**
- **Use "getHost()" and "getFirstMappedPort()" to determine the name+port that JUnit tests can use to access HTTP**
- **Use "getMappedPort(80)" if you mapped more than one port**

```
public String getRootUrl(WebDriver driver) {
    String host;
    int port;
    if(driver instanceof RemoteWebDriver) {
        host = JakartaTestContainers.CONTAINER_NETWORK_NAME;
        port = 80;
    } else {
        host = JakartaTestContainers.instance.domino.getHost();
        port = JakartaTestContainers.instance.domino.getFirstMappedPort();
    }

    String context = getExampleContextPath();
    return PathUtil.concat("http://" + host + ":" + port, context, '/');
}

public String getRestUrl(WebDriver driver) {
    String root = getRootUrl(driver);
    return PathUtil.concat(root, "xsp/app", '/');
}
```

BASIC TESTS

➤ **JAX-RS provides a Client API that works well for basic HTTP requests**

➤ **Tools like Apache HttpClient would work well too**

```
@Test
public void testSample() {
    Client client = getAnonymousClient();
    WebTarget target = client.target(getRestUrl(null) + "/sample");
    Response response = target.request().get();

    String output = response.readEntity(String.class);

    assertTrue(output.startsWith("I'm application guy at"), () -> "Rec
}
```

SELENIUM CONTAINERS

- **Testcontainers provides a Selenium container that can run Chrome or Firefox**
- **The Selenium container can't see the local host name**
- **Give your Domino container a DNS-friendly name**
- **Create a "Network" object using the "bridge" driver**
- **Use "withNetwork" on your Domino and Selenium containers to bind them to this virtual network**
- **https://www.testcontainers.org/modules/webdriver_containers/**

```
public static final String CONTAINER_NETWORK_NAME = "xsp-jakartaee-test";

public final Network network = Network.builder()
    .driver("bridge") //$NON-NLS-1$
    .build();
```

```
domino = new DominoContainer()
    .withNetwork(network)
    .withNetworkAliases(CONTAINER_NETWORK_NAME)
```

```
firefox = new BrowserWebDriverContainer<>()
    .withCapabilities(new FirefoxOptions())
    .withNetwork(network);
```

SELENIUM TESTS

- **Selenium drivers implement `WebDriver`**
- **This provides tools for finding elements, clicking buttons, etc.**
- **The API is generally DOM-like**
- **Elements can be found with XPath, CSS, and a few other mechanisms**
- **Personally, I'm an XPath kind of guy**

```
@ParameterizedTest
@ArgumentsSource(BrowserArgumentsProvider.class)
public void testHelloPage(WebDriver driver) {
    driver.get(getRootUrl(driver) + "/hello.xhtml");

    String expected = "inputValue" + System.currentTimeMillis();
    {
        WebElement form = driver.findElement(By.xpath("//form[1]"));

        WebElement dd = driver.findElement(By.xpath("//dt[text()='Request Method']"));
        assertEquals("GET", dd.getText());

        WebElement input = form.findElement(By.xpath("input[1]"));
        assertTrue(input.getAttribute("id").endsWith(":appGuyProperty"), () -> i
        input.click();
        input.sendKeys(expected);
        assertEquals(expected, input.getAttribute("value"));

        WebElement submit = form.findElement(By.xpath("input[@type='submit']"));
        assertEquals("Refresh", submit.getAttribute("value"));
        submit.click();
    }
    {
        WebElement form = driver.findElement(By.xpath("//form[1]"));

        WebElement span = form.findElement(By.xpath("p/span[1]"));
        assertEquals(expected, span.getText());
    }
}
```

MULTIPLE BROWSERS

- **Use Junit ParameterizedTests to run tests with multiple containers**
- **The Testcontainers image supports Chrome and Firefox**
- **Often, one will suffice, but it's good practice to set this up for when you want to test cross-browser compatibility**
- **You can combine this with non-container drivers (e.g. Safari) if desired**
- **There's an HtmlUnit driver, but it was out of date when I last checked**

```
public class BrowserArgumentsProvider implements ArgumentsProvider {  
  
    @Override  
    public Stream<? extends Arguments> provideArguments(ExtensionContext context) {  
  
        return Stream.of(  
            JakartaTestContainers.instance.firefox  
        )  
        .map(BrowserWebDriverContainer::getWebDriver)  
        .map(Arguments::of);  
    }  
}
```


QUESTIONS?

Use the GoToWebinar Questions Pane

Please keep all questions related to the topics that our speakers are discussing!!!

Unrelated Question => post at:

<http://openntf.slack.com/>

