Title: How to pass secrets to testContainers?

Post Body:

I have following docker-compose file for local development:

```
version: '3.4'  networks:   mynetwork:  services:   samba:     image: instantlinux/samba-dc:latest     container_name: samba-d
```

Now I try to implement integration tests using testContainers for that purpose:

```
@Testcontainers @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT) @ActiveProfiles("test") ....
```

But when I try to run it I receive an error:

```
Container startup failed for image instantlinux/samba-dc:latest .... rg.testcontainers.containers.GenericContainer expected th
```

Looks like I have to configure secrets somehow but I don' see a way how to acheve it.

Update 1

Secret file looks like this:

```
kind: Secret apiVersion: v1 metadata:   name: samba-admin-password data:   ADMIN_PASSWORD_SECRET: superpassword
```

Update 2

Based on VonC answer I've created the example:

```
@Testcontainers @SpringBootTest(webEnvironment = RANDOM_PORT) @ActiveProfiles("test") class TestContainersBase {      @Test
```

In app logs I see:

```
2023-08-21T13:38:50.555+03:00  INFO 15136 --- [    Test worker] o.t.utility.ImageNameSubstitutor         : Image name substitu
```

In docker desktop:

And the first container(based on ports I think that it is Samba) logs:

```
2023-08-21 13:38:58 Set timezone 2023-08-21 13:38:59 INFO 2023-08-21 10:38:59,067 pid:18 /usr/lib/python3.10/site-packages/sam
```

docker ps

```
PS C:\work\myApp\docker> docker ps CONTAINER ID   IMAGE                          COMMAND       CREATED          STATUS           P
```

Accepted Answer:

Testcontainers does not seem to support Docker Compose's secrets directly. The only notion of secret is when using a HashiCorp Vault Module `.withSecretInVault()`.

In your case, you can try and use a volume to emulate a secret: it is a workaround, a volume to bind-mount your secrets into the container at the expected path.

```
val secretPathOnHost = "/path/to/your/secrets.yaml"; val secretPathInContainer = "/run/secrets/samba-admin-password";  val lda
```

Do replace `/path/to/your/secrets.yaml` with the absolute path to your `secrets.yaml` file on the host machine.

Note: Emulating secrets as volumes means the secret is available as a plaintext file on your host system, so make sure you manage its permissions and access properly. That might be fine for local development and testing, but might not be ideal for production-like environments.

And... never commit secrets or secret paths to source control.

Since the secret file is:

```
kind: Secret apiVersion: v1 metadata:   name: samba-admin-password data:   ADMIN_PASSWORD_SECRET: superpassword
```

Given this format, which is similar to a Kubernetes secret, you will need to parse the file in your test setup and then set the secret as an environment variable to the container.

Using `testcontainers` and the Jackson library for YAML parsing:

```
import com.fasterxml.jackson.databind.ObjectMapper; import com.fasterxml.jackson.dataformat.yaml.YAMLFactory; import java.io.F
```

That method involves reading the secret into your Java application, which you then pass as an environment variable to the container.

From the example provided by the OP, it looks like a Samba container using Testcontainers in a Spring Boot application. The container seems to be starting correctly as you can see the initialization logs of Samba.

> The main problem that samba container shutdown immediately after start(application is hanging on in Thread sleep). Looks like root cause could be find in samba logs:
>
> ```
> 2023-08-21 13:39:07 ERROR(runtime): uncaught exception - (3221225506, '{Access Denied} A process has requested access to an object but has not been granted those access rights.')
> ```

Since you are using Docker, remember that the Samba container will have its own user system. You may need to adjust the user or group IDs to match your host system if you are mounting volumes.

Add in your container entry point the `id -a` and `ls -alrth /path/to/your/secrets.yaml` commands to see who you are inside the container, and how you see the mounted file system.
Check the ownership and permissions of any volumes you have mounted into the container. The UID and GID inside the container might differ from those outside, leading to permission issues.

If you are running Samba inside a Docker container, ensure that you have provided all necessary capabilities using `--cap-add` if needed.

Review your Samba configuration (`smb.conf`). Ensure that the shares and paths defined have the correct permissions.
Also, check for any `valid users`, `read list`, or `write list` directives and ensure that the users listed have the appropriate permissions.

And if you are running a system with SELinux enabled, this can cause permission issues. You can temporarily set SELinux to permissive mode to see if it resolves the issue:

```
sudo setenforce 0
```

If this resolves the issue, you will need to create the appropriate SELinux policies or adjust the context for the Samba-related files and directories.

Highest Rated Answer: None