Performance Engineering Home Assignment: JFrog Xray API Load Testing

Role: Senior Performance Engineer (10+ years experience)

Technology Stack: Locust/Python
Duration: 4-6 hours
Submission: GitHub repository

Dear Candidate,

Thank you for your interest in the **Performance Engineer** role at **JFrog**. This assignment evaluates your technical skills, problem-solving abilities, and approach to designing and implementing automation frameworks.

Objective

Design and implement a distributed performance testing framework using Locust and Python to evaluate JFrog Xray's scanning capabilities under load. Your solution should help identify performance bottlenecks and establish baseline metrics for artifact scanning operations.

Please read the documentation thoroughly and tailor your tests accordingly.

Scenario

Using the JFrog Platform APIs, simulate multiple users performing these operations concurrently:

1. Creating Docker repositories
2. Pushing container images
3. Triggering security scans
4. Checking scan status
5. Retrieving violations

Note -

1. **JFrog Artifactory**: A repository manager that stores and organizes software artifacts (like Docker images).
2. **JFrog Xray**: A security tool that scans artifacts for vulnerabilities and licensing issues, acting like a "security guard" for your artifacts.

Prerequisites

To get you started, let’s set up a trial account. Please follow the steps to do so.

Step 1: Trial Account Setup

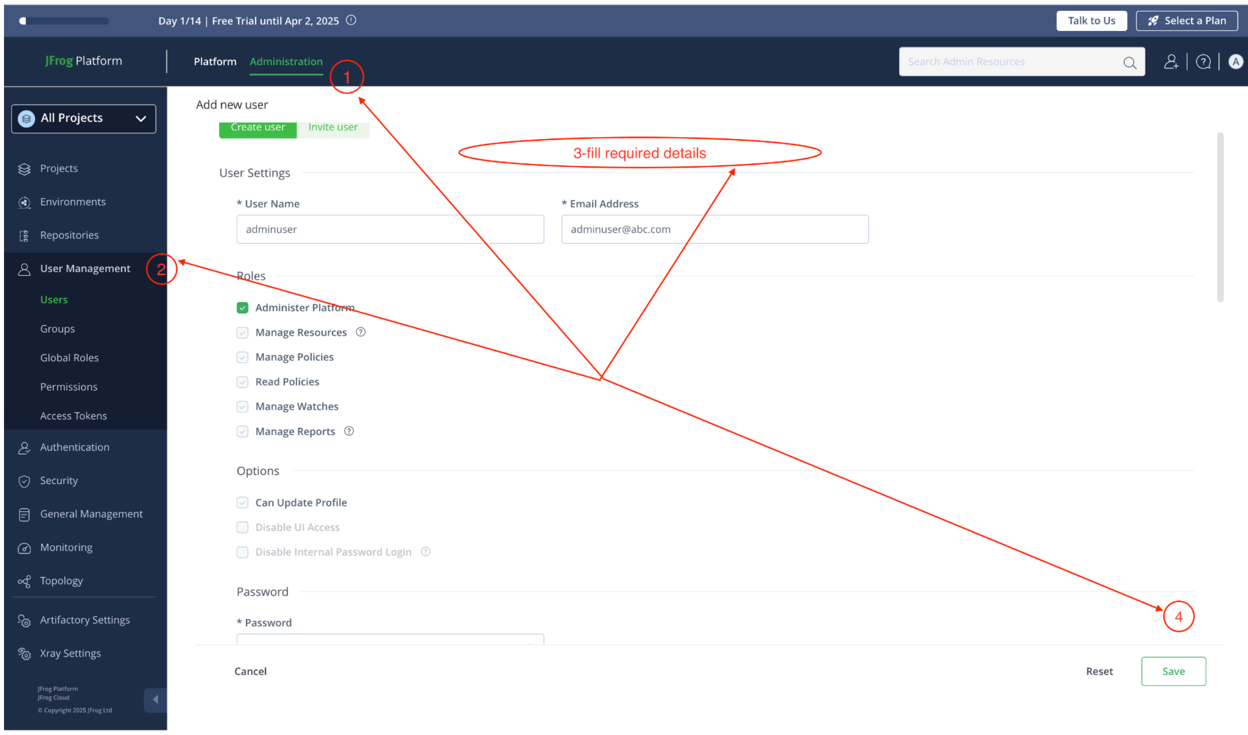
1. Visit the JFrog trial page: https://jfrog.com/start-free/#trialOptions.

2. Select the **14-day trial** option.
3. Choose a signup method (we recommend using Github for simplicity).
4. Follow the instructions to complete the setup.
5. After setup, you'll receive an email from `service@jfrog.com` with your platform details (URL, username, and password).

Please note: If you registered using Github, access the platform by logging in with your Github account and setting up an admin user for this task. Additionally, you will receive a platform URL in the format `https://{platform_id}.jfrog.io`. Take note of the platform_id, as it will be required when pushing a Docker image to the platform.

Step 2: Create an Admin User

1. Log in to the JFrog Platform using your github account.
2. Navigate to the **Administration** tab in the top menu.
3. Click on **User Management** in the left-hand menu.
4. Click **New User** and provide the following details:
   - **Username**: [Choose a username]
   - **Email**: [Provide an email]
   - **Role**: Select **Administer Platform**.
   - **Password**: [Set a password].



User Settings

* User Name: adminuser

* Email Address: adminuser@abc.com

Roles

- [x] Administer Platform
- [ ] Manage Resources
- [ ] Manage Policies
- [ ] Read Policies
- [ ] Manage Watches
- [ ] Manage Reports

Options

- [ ] Can Update Profile
- [ ] Disable UI Access
- [ ] Disable Internal Password Login

Password

* Password

5. Save the user and log out.

6. Log in again using the newly created admin user credentials to check if it works.

Test Implementation

Note :

- **Steps 1-7**: Use REST APIs to create and verify data like repositories, watches, and policies. Each API call includes a **bookmark hyperlink** to its corresponding section for `cURL` requests and responses.
- When setting up the account, the system generates sample data, including policies and watch repositories. It is recommended not to use this sample data for your tasks. You may delete it if needed.

Steps:

Now that your account is set up, let's automate the scenario. Follow these steps in order:

API Steps

1. **Create a Docker Repository**:
   - Use the **Create Repository REST API** to set up a Docker repository.
   - Sample Request and Response
2. **Verify Repository Creation**:
   - Use the **Get Repository REST API** to confirm the repository was created successfully.
   - Sample Request and Response
3. **Push a Docker Image**:
   - Pull the Docker image `alpine:3.9`.
   - Log in to your JFrog Platform using Docker CLI.
   - Tag and push the image to your newly created repository.
   - Example commands:

Unset
```
docker pull alpine:3.9

docker login {your_platform_name}.jfrog.io [no https://]
docker tag alpine:3.9 {jfrog platform url without https://}
{REPO_NAME}/{CUSTOM_IMAGE_NAME}:{CUSTOM_TAG}

docker push {jfrog platform url without
https://}/{REPO_NAME}/{CUSTOM_IMAGE_NAME}:{CUSTOM_TAG}
```

4. **Create a Security Policy**:
   - Use the **Create Policy API** to define a security policy with specific rules.
   - Sample Request and Response
5. **Create a Watch**:
   - Use the **Create Watch REST API** to link the policy and repository.
   - Sample Request and Response
6. **Apply a watch on existing content**:
   - Use the **Apply watch REST API** to apply the watch on a repository created in earlier steps.
   - Sample Request and Response
7. **Check Scan Status**:
   - Use the **Xray REST API** to verify that the image has been scanned.
   - Note that this is an asynchronous operation and will take time. Eventually, you will see `overall": { "status": "DONE" }`
   - Sample Request and Response
8. **Verify Violations**:
   - Use the **Get Violation REST API** to check if violations were generated.
   - Note that this is an asynchronous operation and will take time. Eventually, you will see the `` `total_violations > 0`. ``
   - Sample Request and Response

Distributed Execution

1. Implement master/worker architecture
2. Demonstrate scaling with ≥2 worker nodes.
3. Aggregate results across nodes

Reporting

Generate a report containing:

1. Test configuration parameters
2. Performance metrics with visualizations
3. Observations and recommendations
4. Raw data (CSV/JSON) for further analysis

Deliverables

Please share the following:

1. **GitHub Repository**:
    - Code for the automation framework.
    - A **README** file with setup and execution instructions.
    - Any assumptions or design decisions?
2. **Test Execution Instructions**: Clear steps to run the tests.
3. **Example Report Output** demonstrating:
    - Response time trends under load
    - Throughput capacity
    - Error analysis
4. **Your Analysis** is in a separate file called analysis.md.
    - Post-processing of results for insights or trends

Evaluation

Your solution will be evaluated based on the following:

1. **Framework Design**: Modular, configurable, follows Python best practices
2. **Load Implementation:**. Realistic workload, proper metric collection, and distributed execution
3. **Code Quality**: Must be readable, maintainable, and follow best practices. Remember, you establish the standards for this team.
4. **Error Handling**: Robust handling of failures and graceful degradation.
5. **Analysis & Reporting**: Meaningful metrics, actionable insights, professional presentation

Good luck! We look forward to seeing your approach to performance testing this critical workflow.

Sample API Requests and Responses

Create Repository

Request:

```
curl --location --request PUT
'https://jfsinterview.jfrog.io/artifactory/api/repositories/docker-local' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic bGVhZDpQYXNzd29yZDEh' \
--data '{
   "key": "docker-local",
   "projectKey": "",
   "packageType": "docker",
   "rclass": "local",
   "xrayIndex": true
}'
```

Response:

```
Successfully created repository 'docker-local'
```

Check Repository Existence

Request:

```
curl --location 'https://jfsinterview.jfrog.io/artifactory/api/repositories' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic bGVhZDpQYXNzd29yZDEh' \
--data ''
```

Response:

```
[
  {
   "key": "docker-local",
   "description": "",
   "type": "LOCAL",
   "url": "https://jfsinterview.jfrog.io/artifactory/docker-local",
   "packageType": "Docker"
  }
]
```

Check Scan Status

Request:

```
curl --location 'https://jfsinterview.jfrog.io/xray/api/v1/artifact/status' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic bGVhZDpQYXNzd29yZDEh' \
--data '{
  "repo": "docker-local",
  "path": "/alpine/3.9/manifest.json"
}'
```

Response:

```
{
  "overall": {
    "status": "DONE",
    "time": "2025-02-28T06:41:50Z"
  },
  "details": {
    "sca": {
      "status": "DONE",
      "time": "2025-02-28T06:40:29Z"
    }
  }
}
```

Create Policy

Request:

```
curl --location 'https://mytraial001.jfrog.io/xray/api/v2/policies' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic YWRtaW51c2VyOlBhc3N3b3JkMSE=' \
--data '{
 "name": "sec_policy_1",
 "description": "This is a specific CVEs security policy",
 "type": "security",
 "rules": [
     {
        "name": "some_rule",
        "criteria": {
            "malicious_package": false,
            "fix_version_dependant": false,
            "min_severity": "high"
         },
         "actions": {
            "mails": [],
            "webhooks": [],
            "fail_build": false,
            "block_release_bundle_distribution": false,
            "block_release_bundle_promotion": false,
            "notify_deployer": false,
            "notify_watch_recipients": false,
            "create_ticket_enabled": false,
            "block_download": {
               "active": false,
               "unscanned": false
            }
         },
         "priority": 1
      }
   ]
}'
```

Response:

```
{"info":"Policy has been successfully created"}
```

Create Watch

Request:

```
curl --location 'https://jfsinterview.jfrog.io/xray/api/v2/watches' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic bGVhZDpQYXNzd29yZDEh' \
--data '{
   "general_data": {
      "name": "example1-watch",
      "description": "This is an example watch #1",
      "active": true
   },
   "project_resources": {
      "resources": [
        {
            "type": "repository",
            "bin_mgr_id": "default",
            "name": "docker-local",
            "filters": [
                {
                    "type": "regex",
                    "value": ".*"
                }
            ]
        }
    ]
  },
  "assigned_policies": [
    {
        "name": "sec_policy_1",
        "type": "security"
    }
  ]
}'
```

Response:

```
{"info":"Watch has been successfully created"}
```

Apply a watch to existing content.

Request:

```
curl --location 'https://xray-dev.jfrogdev.org/xray/api/v1/applyWatch' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
--data '{
  "watch_names":[
    "amit-watch-3.117"
  ],
  "date_range":
    {
      "start_date": "2025-04-07T10:25:00+02:00", //UTC time in RFC3339
formatYYYY-MM-DDTHH:MM:SSZ
```

```
     "end_date": "2025-04-07T10:30:00+02:00" // UTC time in RFC3339
formatYYYY-MM-DDTHH:MM:SSZ
   }
}'
```

Response:

```
{
   "info": "History Scan is in progress"
}
```

Get Violations

Request:

```
curl -X POST "https://xray-dev.jfrogdev.org/xray/api/v1/violations" \
-H "Content-Type: application/json" \
-d '{
   "filters": {
      "watch_name": "<$WATCH_NAME>",
      "violation_type": "Security",
      "min_severity": "High",
      "resources": {
         "artifacts": [
            {
               "repo": "${REPO_NAME}", //docker-local
               "path": "${PATH_TO_ARTIFACT/IMAGE}" //alpine/3.9/manifest.json
            }
         ]
      }
   },
   "pagination": {
      "order_by": "created",
      "direction": "asc",
      "limit": 100,
      "offset": 1
   }
}'
```

Response:
```
{
  "total_violations": 5,
  "violations": []
}
```

Good Luck!

We’re excited to see your solution and how you approach this challenge. If you have any questions, feel free to contact us.

Happy coding!