

# SQUEEZE Entwickler Handbuch

Informationen und Dokumentation zur Squeeze Software für Entwickler

- [Aufbau](#)
  - [Worker-System](#)
- [Entwickler-Setup](#)
  - [Einleitung](#)
  - [SSH Key erstellen](#)
- [Tabelleneinstellungen](#)
- [Guidelines](#)
  - [Pathing](#)

# Aufbau

# Worker-System

## Überblick

Dokumente in SQUEEZE werden mit einer Pipeline verarbeitet, einer Sammlung verschiedener Einzelschritte wie z.B. Bildaufbereitung und Texterkennung.

Diese einzelnen Schritte werden dabei von Worker-Prozessen ausgeführt, die unabhängig der API arbeiten.

Ein Worker-Manager koordiniert die Job-Ausgabe und sorgt für eine faire Verteilung bzw. Priorisierung.

Kommt ein neues Dokument in den Verarbeitungsprozess, wird von der SQUEEZE-API ein neuer Job für das Dokument an den Worker-Manager übermittelt. Der nächste freie Worker fordert einen Job vom Worker-Manager an und führt den nächsten anstehenden Pipeline-Schritt aus.

Pro Pipeline-Schritt wird zuerst vom jeweiligen Worker ein Archiv mit den erforderlichen Daten von der API heruntergeladen und in sein lokales Dateisystem übertragen.

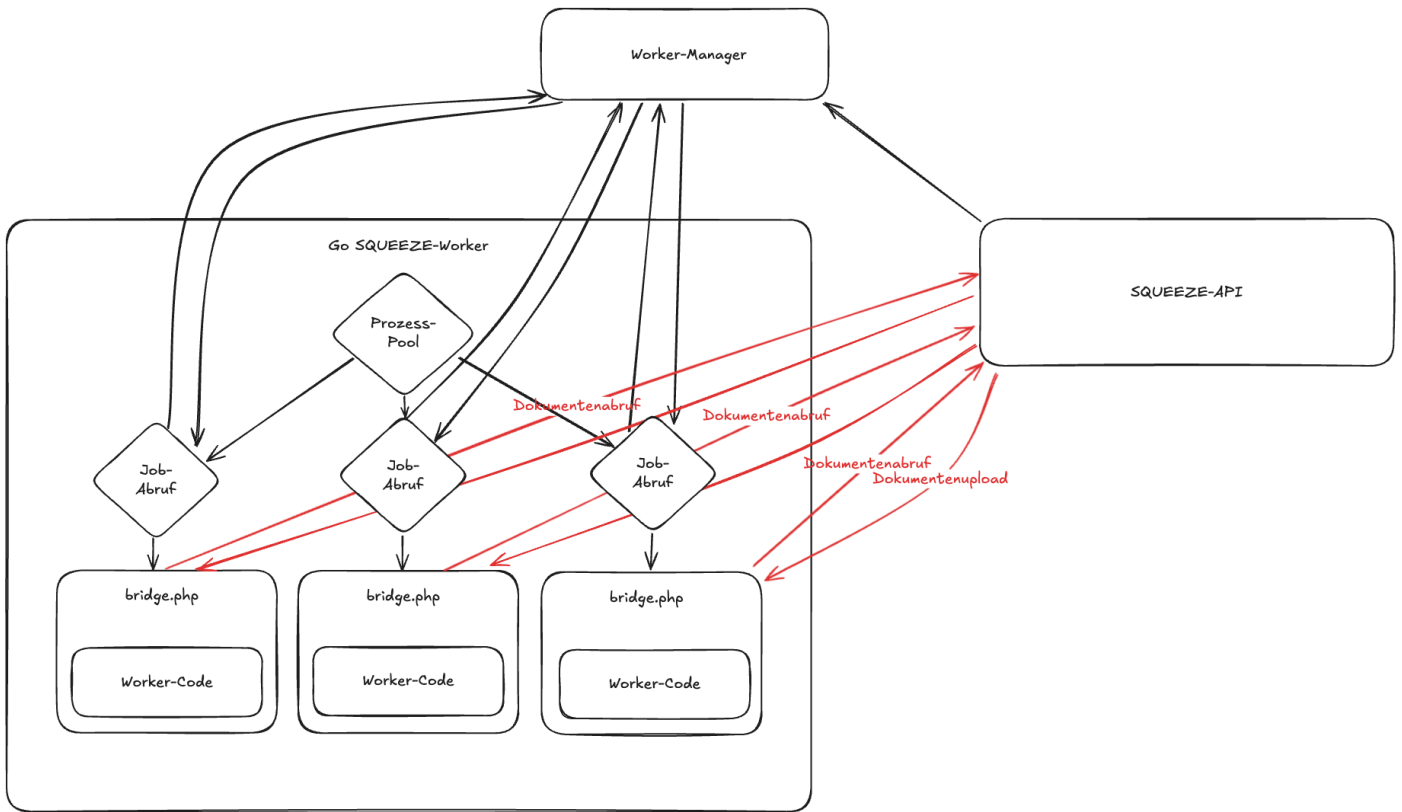
Danach beginnt die eigentliche Verarbeitung, z.B. die Bildaufbereitung oder Extraktion.

Am Ende des Vorgangs wird das Result wieder als Archiv über HTTP zur API hochgeladen.

## Aufbau des Workers

Der Worker und Worker-Manager sind Teil eines einzigen ausführbaren Programmes. Für die Manager-Funktion wird die Executable mit dem zusätzlichen Parameter ``manager`` gestartet.

Worker und Worker-Manager sind in Go geschrieben, die eigentliche Ausführung des Pipeline-Schrittes wird jedoch durch ein PHP-Script (`bridge.php`) bewerkstelligt, was den SQUEEZE-Core lädt und den Schritt ausführt.



# Entwickler-Setup

# Einleitung

Dieses Kapitel soll dir einen Einstieg bieten, um dein lokales Entwickler-Setup aufzusetzen damit du mit der SQUEEZE auf deinem Rechner starten kannst.

Dazu werden folgende Voraussetzungen benötigt:

1. [Docker Desktop installiert](#)
2. [Routing Einrichtung für \\*.localhost \(macOS\)](#) (um squeeze.docker.localhost aufrufen zu können)
3. [Task-Runner installiert](#)
4. [Git installieren](#)
5. [SSH-Key anlegen/in Azure DevOps hinterlegen](#)

## Repository lokal klonen

1. Terminal/CMD öffnen und in einen Ordner deiner Wahl wechseln
2. das SQUEEZE-Repo mit `git clone`  
`git@ssh.dev.azure.com:v3/DEXPRO/DEXPRO%20Platform/SQUEEZE` klonen

Danach kann das Projekt in der IDE z.B. PHPStorm geöffnet werden.

## Docker Registry Login

Um Docker-Images aus unserer Firmen-Registry laden zu können, musst du dich lokal anmelden.

Befolge dazu die Anweisungen in den Abschnitten "Erster Login" und "Einloggen auf der lokalen CLI/im Terminal" auf [dieser Seite](#).

## Initiales Setup

Lege zuerst eine docker-compose.override.yml mit folgendem Inhalt an:

```
services:  
  squeeze:
```

```
logging:
  driver: "local"
  options:
    max-size: "25m"
    max-file: "3"
environment:
#   SQZ_WORKER_MANAGER_URL: "http://host.docker.internal:86"
#   SQZ_WORKER_MANAGER_ACTIVE: "false"
#   CLOCKWORK_ENABLED: "true"
  PHP_IDE_CONFIG: "serverName=Docker"
  XDEBUG_MODE: "debug" # use "off" to temporarily turn of the debugger
  XDEBUG_CONFIG: client_host=host.docker.internal
external_links:
  - az-proxy-laravel.test-1:ocr-proxy
  - portal-laravel.test-1:portal
volumes:
  - ./var/www/html/squeeze
  - ./docker/volumes/repository:/var/www/html/repository:rw
  - ./docker/volumes/logs/backend:/var/www/html/logs:rw
worker:
  environment:
#   SQZ_WORKER_MANAGER_URL: "http://host.docker.internal:86"
#   SQZ_WORKER_MANAGER_ACTIVE: "false"
#   SQZ_WORKER_MANAGER_FORCE: "false"
  SQZ_WORKER_DEBUG: "true"
  PHP_IDE_CONFIG: "serverName=Docker"
  SQZ_USE_TENANT_HOST_FOR_API_CALLS: "1"
  XDEBUG_MODE: "debug" # use "off" to temporarily turn of the debugger
  XDEBUG_CONFIG: client_host=host.docker.internal idekey=Docker
external_links:
  - az-proxy-laravel.test-1:ocr-proxy
  - portal-laravel.test-1:portal
volumes:
  - ./var/www/html/squeeze
  - ./docker/volumes/repository:/var/www/html/repository:rw
  - ./docker/volumes/logs/worker:/var/www/html/logs:rw
squeeze-system-api:
  environment:
  PHP_IDE_CONFIG: "serverName=Docker"
  XDEBUG_MODE: "debug" # use "off" to temporarily turn of the debugger
```

```
XDEBUG_CONFIG: client_host=host.docker.internal
volumes:
  - ./var/www/html/squeeze
  - ./docker/volumes/repository:/var/www/html/repository:rw
  - ./docker/volumes/logs/backend:/var/www/html/logs:rw

mariadb:
  ports:
    - 3306:3306

elastic:
  ports:
    - 9200:9200

rabbitmq:
  ports:
    - 5672:5672

filebeat-squeeze:
  volumes:
    - ./docker/volumes/logs/backend:/mnt/logs/backend:ro
    - ./docker/volumes/logs/worker:/mnt/logs/worker:ro
    - ./docker/volumes/logs/systemapi:/mnt/logs/systemapi:ro

frontend:
  platform: linux/amd64

# traefik:
#   networks:
#     freeze:

#networks:
#   freeze:
#     external: true
#     name: freeze_default
```

Im SQUEEZE-Repository kannst du Squeeze mit den folgenden Commands aufsetzen

1. task up (Startet den docker-compose stack)
2. task setup-server (Richtet eine neue Squeeze Instanz ein)
3. task setup-tenant (Setzt den Tenant squeeze.docker.localhost)

4. docker compose restart worker
5. task brt (Verarbeitet ein Test-Dokument in dem Tenant squeeze.docker.localhost)
  1. Wenn beim `task brt` ein berechtigungsfehler auftritt:

```
git config --global core.fileMode false
mkdir -p ./storage/tmp/squeeze.docker.localhost/Documents
chmod 777 -R ./storage/tmp/squeeze.docker.localhost
```

Das SQUEEZE-Frontend sollte nun unter <http://squeeze.docker.localhost/ui/app/squeeze> erreichbar sein.

Die SQUEEZE V2 Swagger Doc findest du unter <http://squeeze.docker.localhost/api/v2/dist/>.

## SQUEEZE-Stack zurücksetzen

Möchtest du deinen SQUEEZE-Stack für ein Review o.ä. neu aufsetzen, gibt es dafür das Command `task reset`.

Das Command löscht die Container und alle Daten der Docker Volumes, danach werden automatisch die Schritte aus dem Setup oben durchlaufen.

## Weitere Task Commands für die Entwicklung

Im Taskfile.dist.yml im Repository sind weiter nützliche Tasks für die Entwicklung definiert. Du kannst dir diese einfach mit dem Ausführen von `task` im Repo anzeigen lassen:

```
zsh: command not found: task
> task
task: Available tasks for this project:
* brt: Upload the BRT Test file
* build: Build the application
* build-api-clients: Generate API clients
* build-openapi: Generate OpenAPI templates
* build-stubs: Generate PHP stubs
* composer-install: Installs the dependencies written down in composer.json
* create-migration: Generates an PhinxMigration PHP Class.
* down: Stops and removes all containers. Leaves volumes intact.
* format-openapi-attributes: Format OpenAPI PHP attributes (indentation, wrapping, and named-argument order)
* import-sql-backup: Given an SQL backup of a Squeeze tenant, import it into your local database. Use for testing & bug reproduction.
* lint: Run all linters - use this to ensure that the combination of all linters is happy.
* lint-migrations: Check if all Phinx migration files follow the naming convention.
* phpstan: Run phpstan on the code base
* phpunit: Run all phpunit testsuites
* phpunit-cli: Run CLI testsuite
* phpunit-e2e: Run E2E testsuite
* phpunit-integration: Run Integration testsuite
* phpunit-rector: Run Rector testsuite
* phpunit-unit: Run Unit testsuite
* psalm: Run psalm on the code base
* psalm-reduce: Reduce the psalm baseline. Use if you have fixed errors.
* psalm-set: Set the psalm baseline. Use if you have added errors. You probably shouldn't ...
* purge: Stops the application and removes all data. Dangerous!
* rector: Run rector which refactors code accordingly to defined rules
* remove-v2-ui: Removes any v2 UI files from the public/ui directory so you can add a new build of the frontend.
* reset: Stops the application, deletes all data and restarts everything
* setup-api-tests: Prepares API tests for running. Use this if you must use a local build of the api client when running tests.
* setup-e2e-testing: Set up your local testing to allow the execution of E2E tests
* setup-server: Sets up a server
* setup-tenant: Sets up tenant
* strict-check: Check if php files declare strict type
* test-api: Run API tests on a running system
* up: Start and update docker containers. Uses some default profiles.
* update: Stops and removes all containers. Leaves volumes intact. Pulls new images and restarts everything.
task: Task "default" does not exist
```

# SSH Key erstellen

Um Zugriff auf die Azure-Repos zu erhalten, benötigst du einen SSH-Key Pair (Private Key / Public Key).

Wenn du schon einen hast, überspringe den ersten Teil.

## SSH-Key erstellen

1. Stellt zu erst sicher, dass das ``.ssh`` Verzeichnis existiert: ``mkdir $HOME/.ssh``
2. Dann legt ihr einen neuen Key an: ``ssh-keygen -t rsa``. Beachte, dass der Typ zwingend ``rsa`` sein muss, dies ist der einzige Typ der von Azure DevOps unterstützt wird. Bestätige die Standard optionen.

## Public-Key auslesen

1. Mit dem Befehl ``cat ~/.ssh/id_rsa.pub`` den public key auslesen.
2. Unter euren Benutzereinstellungen in Azure den Public-Key hinzufügen  
[https://dev.azure.com/DEXPRO/\\_usersSettings/keys](https://dev.azure.com/DEXPRO/_usersSettings/keys)
3. Damit solltest du mit git clone Zugriff auf Azure haben

# Tabelleneinstellungen

# Guidelines

Rules to live by

# Pathing

```
Path::join($absolutePath, xDocFolders::INPUT, 'intermediate.xml')  
$absolutePath . "Input" . DIRECTORY_SEPARATOR . $file  
$absolutePath . xDocFolders::INPUT . DIRECTORY_SEPARATOR . "xrechnung.pdf"  
$absolutePath . "Input" . $dirSep . $file
```

verschiedene Varianten einen Pfad zu verwenden bzw. zu generieren

dabei ist in Zukunft nur

```
Path::join($absolutePath, xDocFolders::INPUT, 'intermediate.xml')
```

zu verwenden, da "diese sich um den ganzen Scheiß mit dem PATH\_SEPARATOR selbst kümmert"  
(Zitat Flo :)