# Mermaid

- docker-access
- repo-migration-issue-index
- Admin Sets Instance Message For First Time
- Saving confirmation
- Service Message Shown At Login
- threat-analysis-remote-login-propagation
- open-id-connect-flow

# docker-access

```
sequenceDiagram
    Runner->>Docker: Start job container
    create participant Ubuntu as Job Container
    Docker->>Ubuntu: Create container
    Docker->>Runner: Container `abcdef` started
    Runner->>Docker: Run `docker build` in `abcdef`
    Docker->>Ubuntu: Run `docker build`
    Note left of Ubuntu: `docker` CLI within the container<br/>will attempt to access a running<br/>docker daemon, by default at <br/>`/var/run/docker.sock`
    Ubuntu-xUbuntu: Open /var/run/docker.sock
    rect rgb(255, 225, 225)
    Note left of Ubuntu: As nothing in the container is<br/>listening to `/var/run/docker.sock`,<br/>the command will fail
    Ubuntu-xDocker: Failed to run `docker build`!
    Docker-xRunner: Failed to run<br/>`docker build` in `abcdef`!
    end
```

```
sequenceDiagram
    create participant Runner
    create participant Docker
    Note over Docker: Running Forgejo as<br/>containerized service
    Runner->>Docker: Start job container
    create participant Ubuntu as Job Container
    Docker->>Ubuntu: Create container
    Docker->>Runner: container `abcdef` started
    Runner->>Docker: Run `docker exec forgejo ...` in `abcdef`
    Docker->>Ubuntu: Run `docker exec forgejo ...`
    Note over Docker,Ubuntu: If the `docker` CLI within the<br/>container can access the `Docker` node,<br/>then a workflow's job running here will<br/>be able to run commands like `docker ps`<br/>to view the other containers,<br/>`docker inspect` to access their environment<br/>variables, and `docker exec` to start<br/>commands within those containers.
    Ubuntu-xDocker: Open /var/run/docker.sock<br/>and exec in forgejo container
```

```
sequenceDiagram
    Runner->>Docker: Start job container
```

```
    create participant Ubuntu as Job Container
    Docker->>Ubuntu: Create container
    Docker->>Runner: container `abcdef` started
    Runner->>Docker: Run `docker build`<br/>in `abcdef`
    Docker->>Ubuntu: Run `docker build`
    Ubuntu->>DIND: Build container...
    DIND->>Ubuntu: Container built!
    Ubuntu->>Docker: Success!
    Docker-xRunner: Success!<br/>Next step please!
```

```
sequenceDiagram
    Runner->>Docker: Start job container
    create participant Ubuntu as Job Container
    Docker->>Ubuntu: Create container
    Docker->>Runner: container `abcdef` started
    Runner->>Docker: Run `docker kill forgejo`<br/>in `abcdef`
    Docker->>Ubuntu: Run `docker kill forgejo`
    Ubuntu->>DIND: Kill container forgejo
    DIND->>Ubuntu: no such container
    destroy Ubuntu
    Ubuntu-xDocker: Failed to run<br/>`docker kill forgejo`!
    Docker-xRunner: Failed to run<br/>`docker kill forgejo`<br/>in `abcdef`!
```

```
sequenceDiagram
    Runner->>Docker: Start job container
    create participant Ubuntu as Job Container
    Docker->>Ubuntu: Create container<br/>Share /var/run/docker.sock
    Docker->>Runner: container `abcdef` started
    Runner->>Docker: Run `docker build` in `abcdef`
    Docker->>Ubuntu: Run `docker build`
    Note left of Ubuntu: `docker` CLI within the Ubuntu<br/>container will attempt to access a
running<br/>docker daemon, by default at <br/>`/var/run/docker.sock`
    Ubuntu->>Ubuntu: Open /var/run/docker.sock
    Ubuntu->>Docker: Redirected /var/run/docker.sock
    Ubuntu->>Docker: Run `docker build`
    Docker->>Ubuntu: [Build Result]
    Ubuntu->>Docker: Running `docker build`: [Command Result]
    Docker->>Runner: Complete: [Command Result]
```

```
sequenceDiagram
    box Runner Host
    participant Runner
    participant LXC
    end
    Runner->>LXC: Start job container
    create participant Debian as Job Container
    LXC->>Debian: Create container
    create participant Docker
    Debian->>Docker: Startup
    LXC->>Runner: container `abcdef` started
    Runner->>LXC: Run `docker build` in `abcdef`
    LXC->>Debian: Run `docker build`
    Debian->>Docker: Open /var/run/docker.sock
    Debian->>Docker: Run `docker build`
    Docker->>Debian: [Build Result]
    Debian->>LXC: Running `docker build`: [Command Result]
    LXC->>Runner: Complete: [Command Result]
    box LXC Container
    participant Debian
    participant Docker
    end
```

# repo-migration-issue-index

```
sequenceDiagram
    participant migrations/migrate.go
    participant github.go
    participant gitea_uploader.go

    migrations/migrate.go->>migrations/migrate.go: migrateRepository(..., downloader,
uploader, ...)
    migrations/migrate.go->>github.go: downloader.SupportGetRepoComments()
    github.go-->>migrations/migrate.go: 'true'
    alt opts.Issues - similar for opts.PullRequests
        migrations/migrate.go->>github.go: downloader.GetIssues()
        migrations/migrate.go->>gitea_uploader.go: uploader.CreateIssues()
    end

    alt opts.Comments && supportAllComments
        migrations/migrate.go->>github.go: downloader.GetPullRequests()
        loop i
            migrations/migrate.go->>github.go: downloader.GetAllComments(i, commentBatchSize)
            github.go->>github.go: g.getClient().Issues.ListComments(...)
            alt check if only PRs or Issues should be migrated
                github.go->>github.go: g.filterPRComments(comments) // Case no PRComments
should be present
            end
            migrations/migrate.go->>gitea_uploader.go: uploader.CreateComments(comments...)
        end
    end
```

# Admin Sets Instance Message For First Time

```
sequenceDiagram
    participant adm as Admin
    participant front as Forgejo Frontend
    participant back as Forgejo Backend

    adm ->> front: Navigate to http://forgejo.instance/admin/service_message (GET)
    front ->> back: Call admin.GetServiceMessage with ctx
    back ->> back: Fill ctx with data
    back ->> front: render tplServiceMessage with ctx data
    front ->> adm: Serve Admin Service Message Management Site
    adm ->> front: Enter Text "Lorem Ipsum" in Modal Service Message field
    adm ->> front: Click "Save"
    front ->> front: Convert to form data so backend can extract
    front ->> back: POST form data to {{AppSubUrl}}/admin/service_message?sm_type=modal
    back ->> back: Route request to CreateOrUpdateServiceMessage web endpoint
    back ->> back: Parse Data with binding ServiceMessageForm{}
    back ->> back: Get value for key "sm_type" (modal)
    back ->> back: Create "Service Message" with text and type "modal" in DB
    back ->> front: Redirect to http://forgejo.instance/admin/service_message
    front ->> adm: Show Updated text field holding entered text
```

# Saving confirmation

```
sequenceDiagram
    participant usr as New User
    participant front as Front End
    participant back as Back End

    usr ->> front: Login
    front ->> back: Send Login Data
    back ->> back: Create ctx
    back ->> back: Check Credentials and Authorize the login
    back ->> back: Check for existence of ServiceMessage by type
    alt service message of type modal exists
        back ->> back: User.MustShowServiceMessages(smType, smTimestamp)
        alt service message needs to be shown
            back ->> back: Set ctx.Data["ModalServiceMessageTitle"]
            back ->> back: Set ctx.Data["RenderedContent"]
        end
    end
    back ->> front: Redirect to dashboard
    front ->> front: Check if ModalServiceMessageTitle is set
    alt modal service message is set
        front ->> usr: Serve html rendered Service Message Modal
        usr ->> front: Read and click "Confirm" button
        front ->> back: POST confirmation to {{AppSubUrl}}/user/confirm?sm_type={{.Type}}
        back ->> back: User.SetConfirm(sm.Type)
        back ->> back: Unset ctx.Data["ModalServiceMessageTitle"]
    end
    front ->> usr: Redirect to initially visited page
```

# Service Message Shown At Login

```
sequenceDiagram
    participant adm as Admin
    participant usr as New User
    participant repo as Forgejo Instance

    adm ->> repo: Navigate to http://forgejo.instance/admin/service_message
    adm ->> repo: Paste Community Guidelines Markdown Formatted text into "Modal Service
Message" text field
    adm ->> repo: Click Save
    usr ->> repo: Login
    repo ->> repo: Check if user has confirmed service message
    alt user has not confirmed
        repo ->> usr: Serve html rendered Service Message Modal
        usr ->> repo: Click "Confirm" button
        repo ->> repo: Save timestamp of accept in user model
    end
    repo ->> usr: Serve dashboard
```

# threat-analysis-remote-login-propagation

```
sequenceDiagram
    participant fs as foreign_repository_server
    participant os as our_repository_server

    fs ->> os: post /api/activitypub/repository-id/1/inbox {Like-Activity}
    activate os
    os ->> repository: load "1"
    os ->> os: validate actor id inputs
    activate os
    os ->> FederationInfo: get by Host
    os ->> os: if FederatonInfo not found
    activate os
    os ->> fs: get .well-known/nodeinfo
    os ->> NodeInfoWellKnown: create & validate
    os ->> fs: get api/v1/nodeinfo
    os ->> NodeInfo: create & validate
    os ->> FederationInfo: create
    deactivate os
    os ->> ForgeLike: validate
    deactivate os

    os ->> user: search for user with actor-id
    os ->> os: create user if not found
    activate os
    os ->> fs: get /api/activitypub/user-id/{id from actor}
    os ->> ForgePerson: validate
    os ->> user: create user from ForgePerson
    deactivate os
    os ->> repository: execute star
    os ->> FederationInfo: update latest activity
    os -->> fs: 200 ok
    deactivate os
```

```
flowchart TD
    A(User) --> |stars a federated repository| B(foreign repository server)
    B --> |Like Activity| C(our repository server)
    C --> |get NodeInfoWellKnown| B
    C --> |get NodeInfo| B
    C --> |get Person Actor| B
    C --> |cache/create federated user locally| D(our database)
    C --> |cache/create NodeInfo locally| D(our database)
    C --> |add star to repo locally| D
```

# open-id-connect-flow

```
sequenceDiagram
    autonumber
    participant Workflow Run
    participant Forgejo
    participant External Platform

    Workflow Run->>Forgejo: Request OIDC ID token
    activate Forgejo
        Forgejo-->>Workflow Run: Generate and return OIDC ID token
    deactivate Forgejo
  Workflow Run->>External Platform: Exchange OIDC ID token for external credentials
    activate External Platform
        External Platform-->>Forgejo: Request OIDC metadata and signing keys
        activate Forgejo
            Forgejo-->>External Platform: Respond with OIDC metadata and signing keys
        deactivate Forgejo
        External Platform-->>External Platform: Verify OIDC ID token signature and claims
        External Platform-->>Workflow Run: Return credentials
    deactivate External Platform
```