sumup®

# OIDC @ SumUp

## w/ Hydra & Terraform

Stepan Rakitin, Platform Engineering

sumup®

**What we do:**

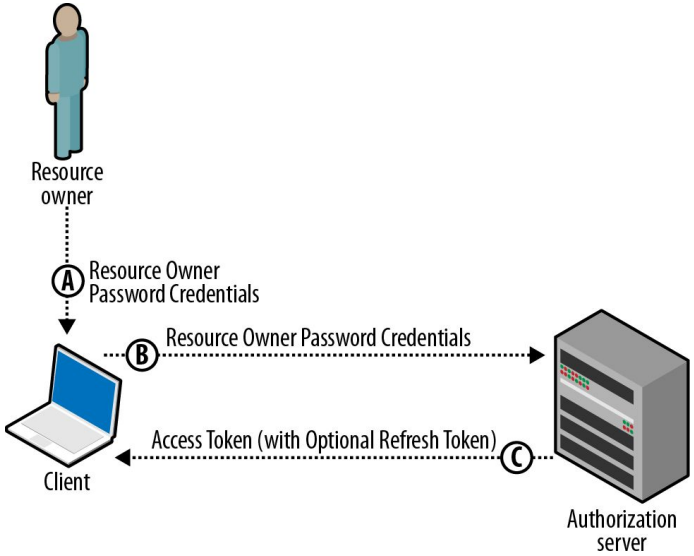
- Acquiring w/ Card Terminals
- Banking
- Accounting & Invoicing
- Online Store
- Developer APIs (Online Payments)

# Agenda

- OAuth 2.0 & Legacy
- SSO w/ Hydra & OIDC
- Self-service w/ Terraform
- Tips & Tricks

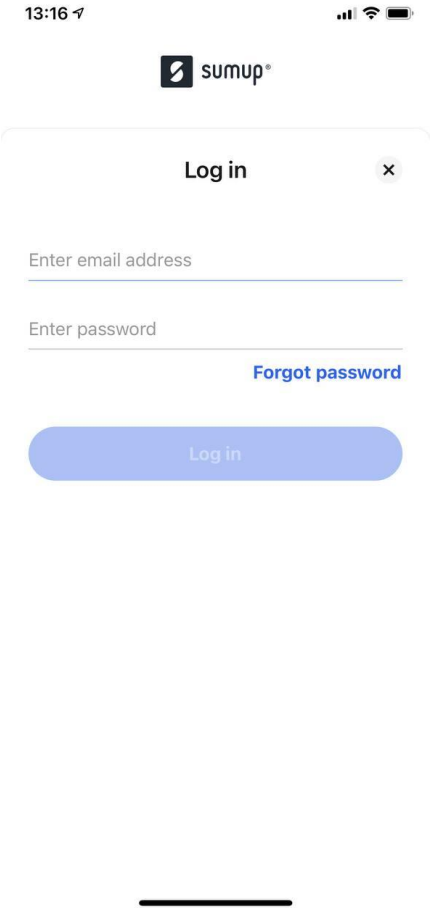
# Legacy

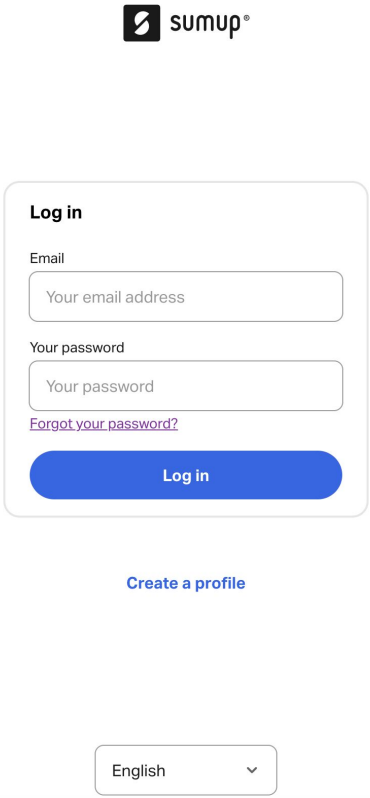
- Our own OAuth 2.0 server
  - Resource Owner Password Credentials (ROPC)
  - Token Exchange (rfc8693)
  - Authorization Code Flow and Client Credentials
- Authorization via Token Introspection



# Problem: Authentication

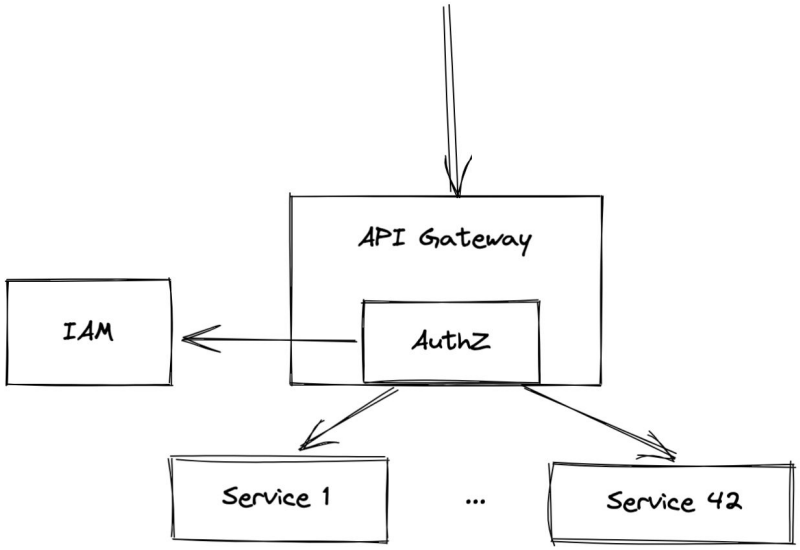
- Authentication is decentralized
  - Many login pages
  - Multiple entrypoints to protect
  - ROPC is insecure and deprecated
  - Difficult to make changes
- No shared session between apps
  - Sharing happens through reauthentication or passing access tokens around :/



# Problem: Authorization

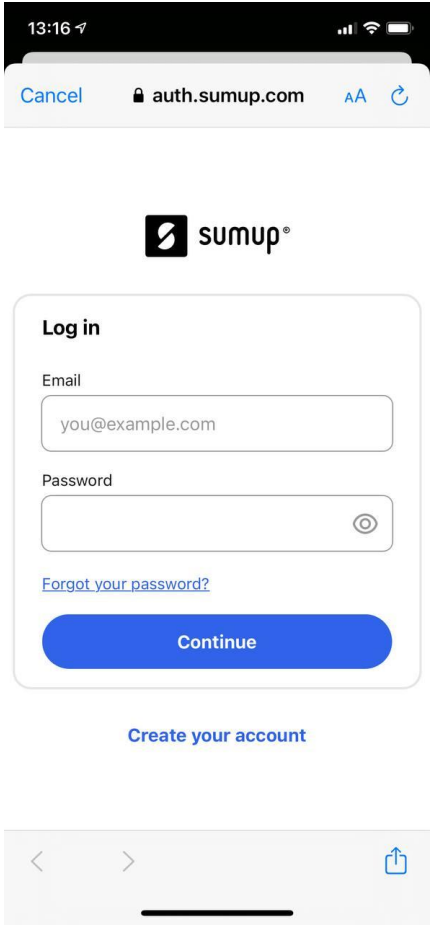
- Authorization is centralized
  - Tokens are opaque
  - Tokens are verified by the central API Gateway
- Trust
  - Need to ensure that traffic only goes through designated gateways to prevent side-attacks

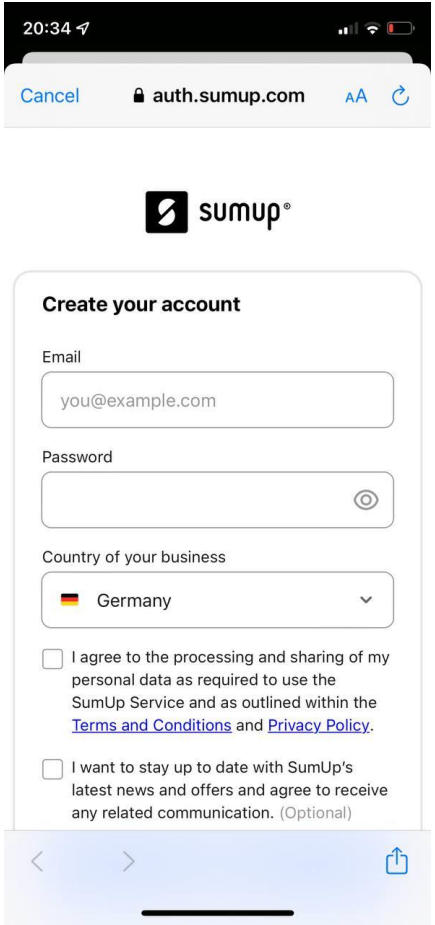


# SSO & OIDC

## Authentication

- Authentication is centralized
  - Single login/signup page to protect
  - Auth Code Flow w/ PKCE
  - ID Tokens carry profile data
  - Trigger MFA via ACR

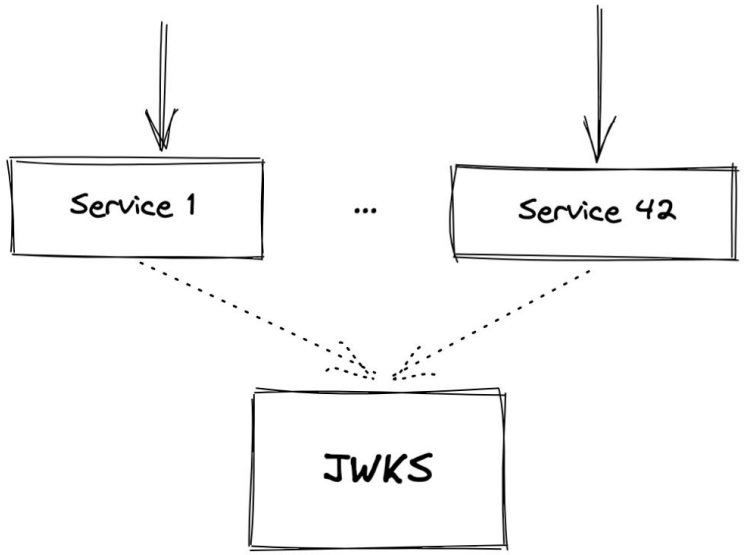


# SSO & OIDC

## Authorization

- Authorization is decentralized
  - Access tokens are JWT
  - Keyset (JWKS) can be used by any client to verify tokens (even outside SumUp)
  - Zero-trust

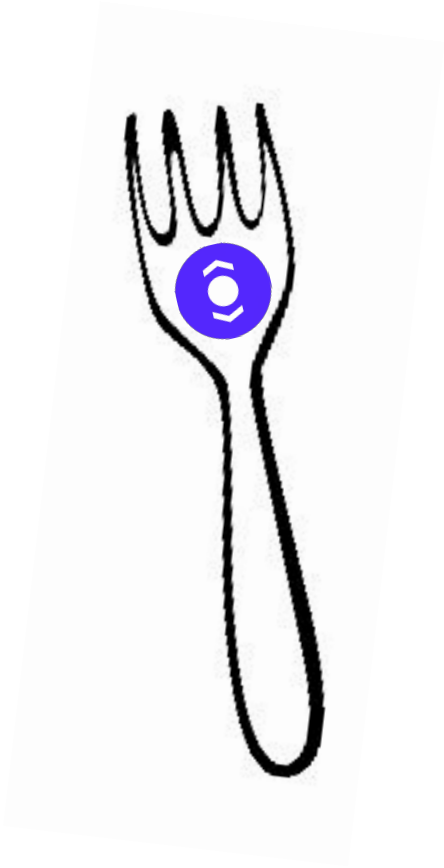


# Hydra: Deployment

- HA k8s deployment (AWS EKS)
  - Managed by jsonnet & grafana/tanka
- Postgres (AWS RDS)
- Protected by Cloudflare

# Hydra: Setup

- We run a fork
  - Per client token TTLs
  - openid-connect-prompt-create-1_0
  - Refresh Token Reuse Detection (upstream)
  - Refresh Token Hook (upstream)
  - Admin server w/o TLS (upstream)
  - Other QoL changes to ease migration
- Authentication app is a mix of Golang & TypeScript



# Hydra: Migration

- Too many clients
  - No direct access to Admin API
  - Teams should be able to self-service
- ROPC -> Authorization Code Flow w/ PKCE
  - Native apps and SDK have to show SSO in the browser
- Different grant types
  - Some of them are unsupported by Hydra (e.g. token exchange)
- Different token auth methods

# Terraform (almost) everything

HashiCorp Terraform

Write, Plan, and Create Infrastructure as Code

- Can be used to manage any resources with CRUD lifecycle, not just cloud infrastructure
- GitOps
  - Resource is modified through PR
  - Change is planned
  - Change is tested
  - Change is applied
  - Rollbacks are easy
- Except frequently changing things (e.g. application releases)

# Managing Hydra resources with Terraform

- terraform-provider-hydra
  - hydra_oauth2_client
  - hydra_jwks
- k8s CRD alternative - ory/hydra-maester
  - Doesn't integrate well with anything non-k8s
  - Templating as a way to reuse code (Helm, jsonnet)

# Terraforming OAuth 2.0 clients

```
# oauth2_client.tf

resource "hydra_oauth2_client" "example" {
  client_id   = "example"
  client_name = "example"

  redirect_uris = [var.example_redirect_uri]
  scopes        = concat(local.default_scopes, ["profile"])

  token_endpoint_auth_method = "client_secret_post"
}


resource "vault_generic_secret" "example_client_credentials" {
  path = "secret/example/client_credentials"

  data_json = jsonencode({
    client_id     = hydra_oauth2_client.example.client_id
    client_secret = hydra_oauth2_client.example.client_secret
  })
}
```

# Terraforming OAuth 2.0 clients

```
# oauth2_client.tf

resource "hydra_oauth2_client" "example" {
  ...
}


resource "kubernetes_secret" "example_client_credentials" {
  metadata {
    name      = "example-client-credentials"
    namespace = "example"
  }

  data = {
    client_id     = hydra_oauth2_client.example.client_id
    client_secret = hydra_oauth2_client.example.client_secret
  }
}
```

# Terraforming JWKS

```
# jwks.tf

resource "hydra_jwks" "id_token" {
  name = "hydra.openid.id-token"

  generator {
    alg = "RS256"
    kid = "generated"
    use = "sig"

    keepers = {
      timestamp = resource.time_rotating.jwks_rotation.id
    }
  }
}


resource "time_rotating" "jwks_rotation" {
  rotation_days = 30
}
```

# Generate your own client credentials

```
# oauth2_client.tf

resource "random_string" "example_client_id" {
  length = 32
}

resource "random_password" "example_client_secret" {
  length = 32
}

resource "hydra_oauth2_client" "example" {
  client_id     = random_string.example_client_id.result
  client_secret = random_password.example_client_secret.result

  ...
}
```

# Use client metadata to customize behavior

```
# oauth2_client.tf

resource "hydra_oauth2_client" "example" {
  ...

  metadata = {
    # Scopes are granted without explicit request
    "default_granted_scopes" = "profile"

    # Automatically accept the consent on behalf of user
    "skip_consent" = true

    # Disable a session, forcing user to always login
    "skip_session" = true

    # Set default max age for a session
    "default_session_max_age" = "15m"
  }
}
```

# Extract common patterns into modules

```
# main.tf

module "public_client" {
  source = "./modules/oauth2_client

  public = true
}

output "public_oauth2_client_credentials" {
  value = {
    client_id = module.public_client.client_id
  }
}

module "secret_client" {
  source = "./modules/oauth2_client

  token_endpoint_auth_method = "client_secret_basic"
}

output "public_oauth2_client_credentials" {
  value = {
    client_id     = module.secret_client.client_id
    client_secret = module.secret_client.client_secret
  }
}
```

# Q&A

https://github.com/svrakitin
https://www.linkedin.com/in/svrakitin

https://auth.sumup.com/.well-known/openid-configuration