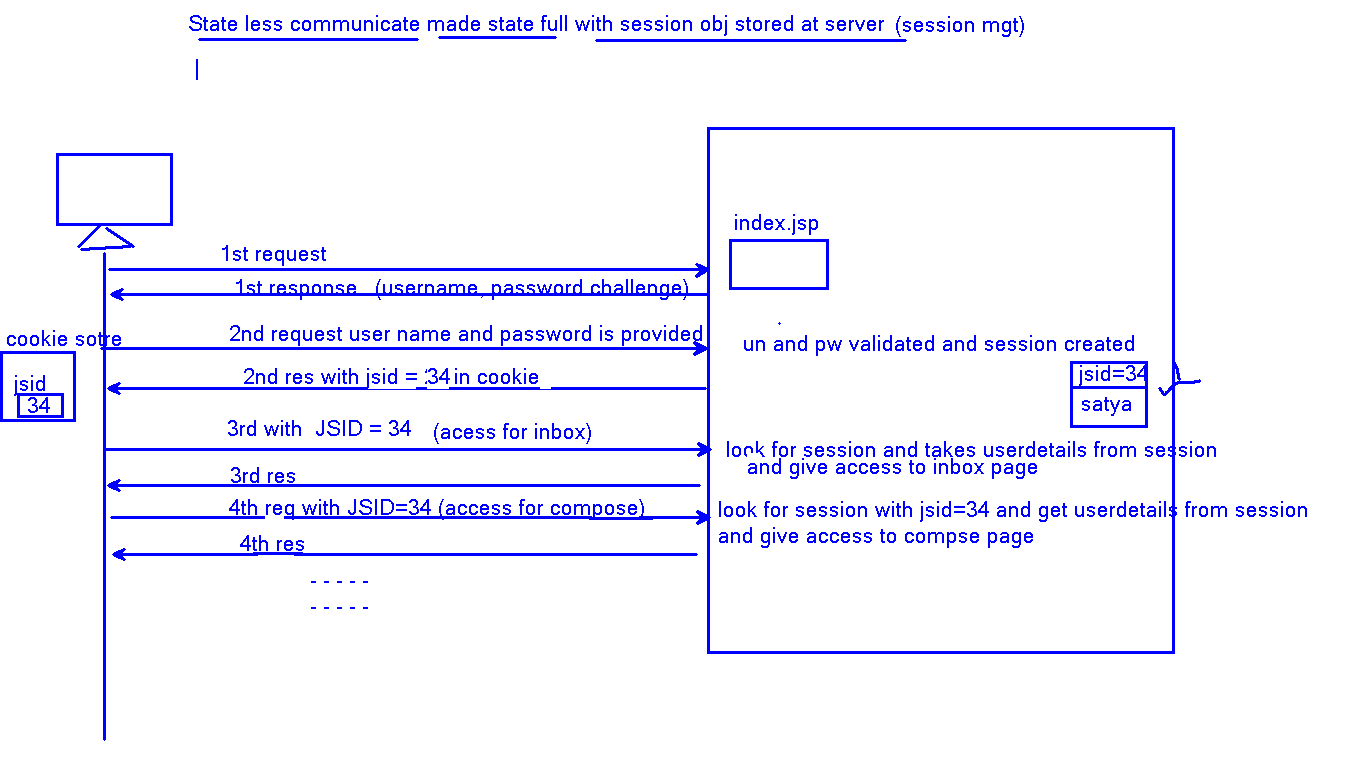
**Http is a state less protocol**

To add state management along with identity management we use cookie + http sessions.



**Session based authentication process (cookie + session)**

1. browser will request for access a resource

2. server side component will verify whether you have access to it or not by checking whether you have session or not

if no session then depending on authentication type:

send challenge for username and password, face recognition, thumb recognition, OTP, certificate, multi factor based attention etc. and goto step3

if there is a session get the user principle (logged in user) from the session goto step5

3. Take the username and password input from browser

check the authentication (by verifying the user name and password with DB/LDAP/AD)

if valid user create a session and return the session id to browser in form of cookie

4. browser will store the session id cookie in cookie store

from now on words browser sends the requests to server with the cookie having session id.

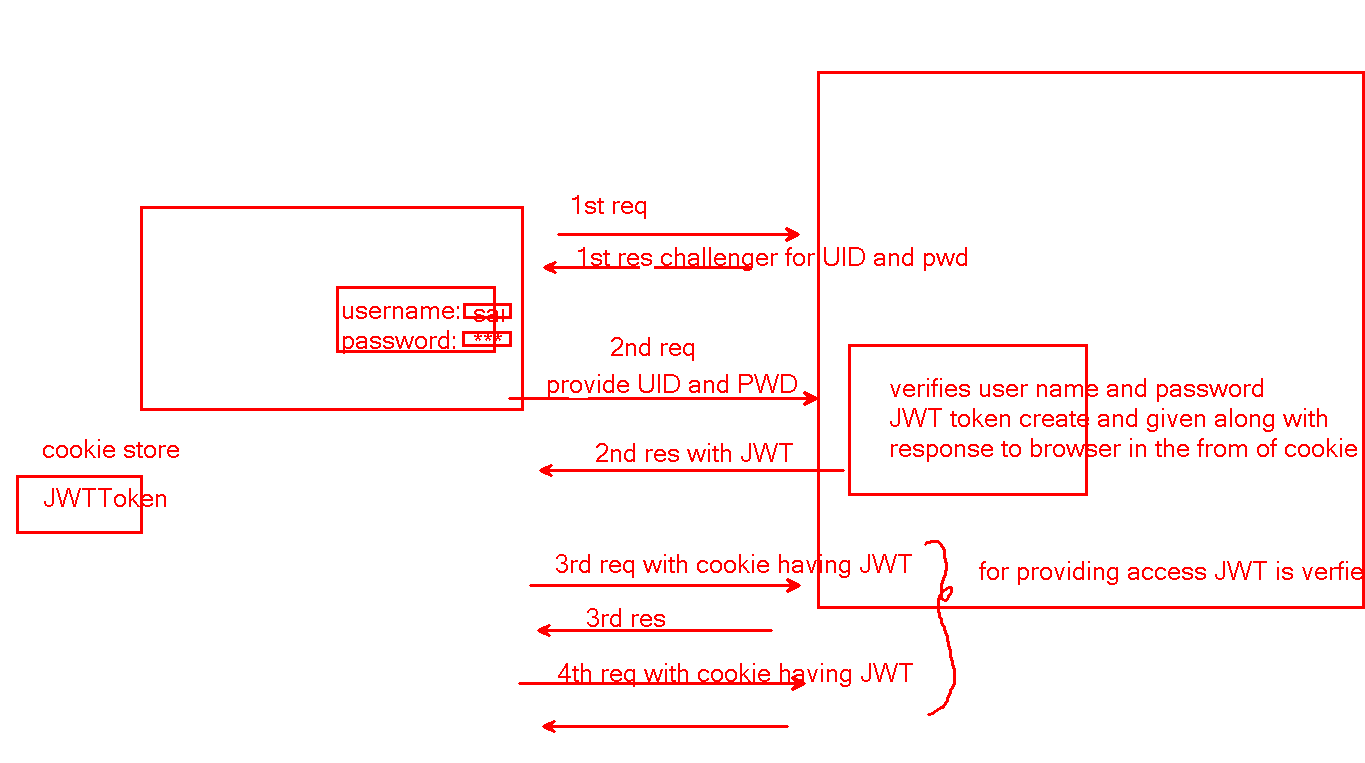
5. Server will use the cookie for identity management (ie for authorization)

**Session less authentication process(JWT tokens, OAUTH, SAML)**

Same as above but no session is created at server side.

in browser cookie JWT token will be stored.

for giving access to the resource server will verify the JWT token.



example use cases where JWT toke is used:

1. Webservices Authorization

2. Horizontal Auto Scaling Web applications,

Scaling Microservices

3. Resource Grant (Open Authorization/OAuth)

Register/Login + Login with Google, Facebook

pros:

It is best suits for Distributed Applications (example micro services)

As Session less so by default token is not stored at server.

If you want to store token at server side for session implementation, we can use In memory cache/DB

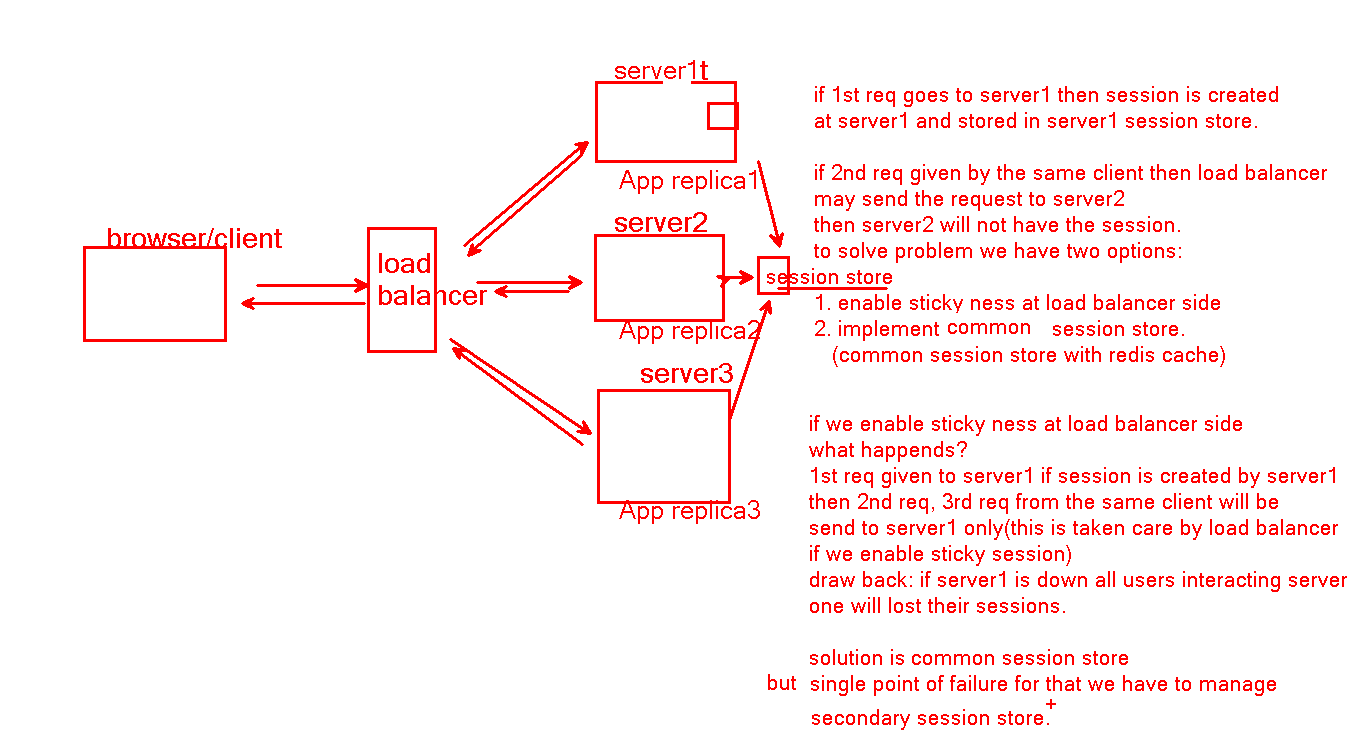
Cons:

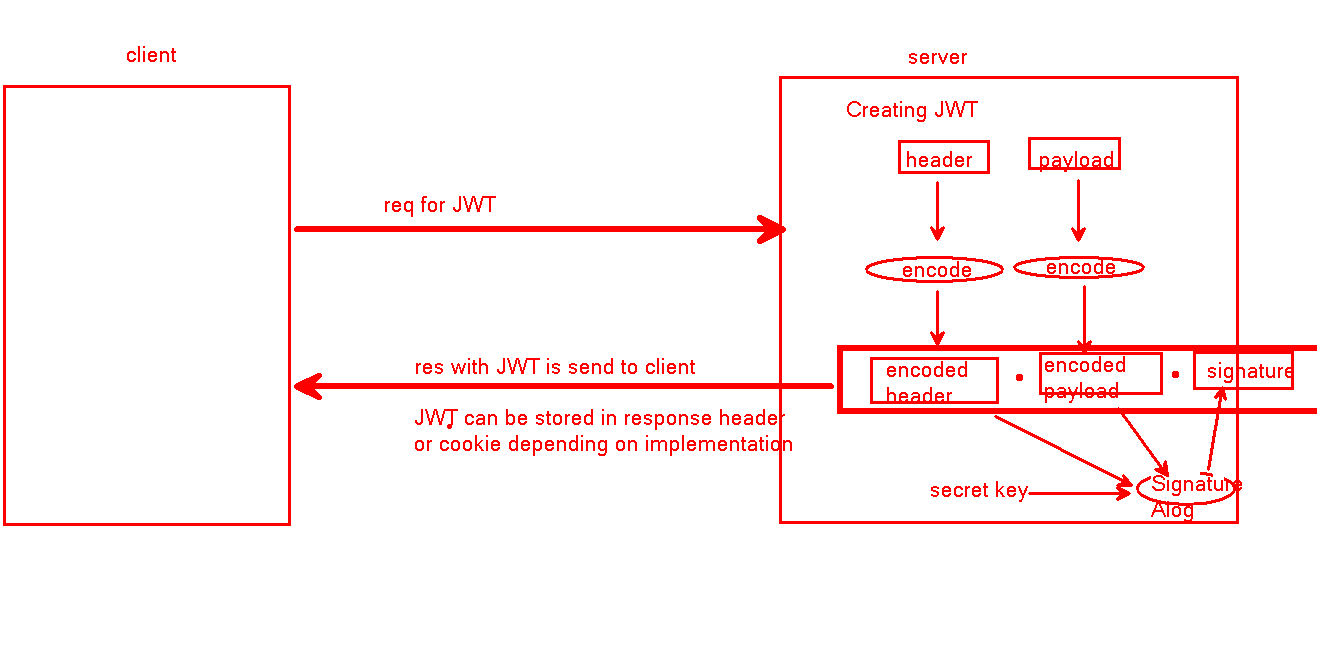
Entire Token must be pass to server on every request. Entire token is validated at server side.

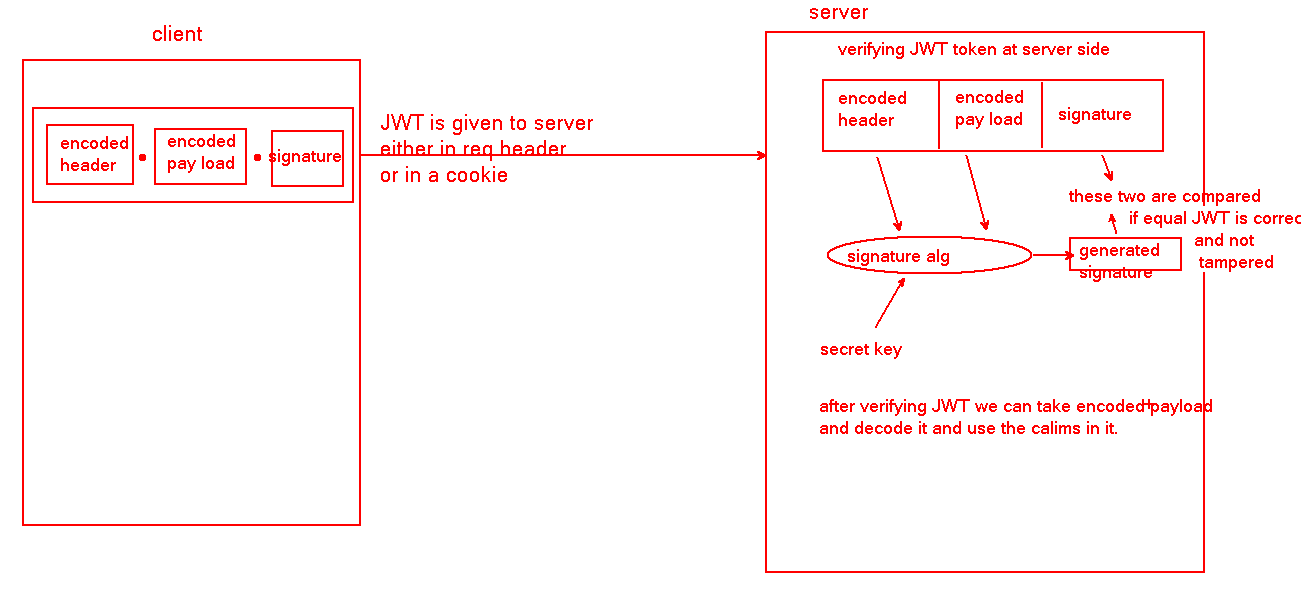
Security breach: If Token is shared with others then they can access client data/services.

Horizontal Auto scaling web applications will use load balancer.

If the web application's state and identity management is developed by using http sessions then load balancer side, we have to enable sticky sessions. Otherwise User has to login multiple times.







JWT token has three parts (JSON Web Token)

1.Header : Base64 encoded JSON

2.Payload : Base64 encoded JSON

3.Signature: hashfunction(Base64 Encoded(Header+Payload)) with a secretkey

example of Header with out encoding:

{

"alg": "HS256",

"typ": "JWT"

}

example of Pay load with out encoding:

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022

}

example of signature:

HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload),a secretkey )

above header after encoding:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

above payload after encoding:

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ

singnature of the header and pay load:

SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

total JWT is:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

JSON Web Tokens are an open, industry standard "RFC 7519" method for representing claims securely between two parties.

or

Means: Two software applications can communicate with each other securely by using JWT tokens.

or

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.

(JWT method of communication make sures that data integrity)

in this method data temparing is not possible. if data is tampared we can identify it.

(until secret key used for signature generation is kept secret)

JSON Web Signature Structure

[Based 64 encoding of Header] . [Base 64 encoding of payload] . [signature]

The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

Very important terminology

--------------------------

The JSON object consists of zero or more name/value pairs (or members), where the names are strings and the values are arbitrary JSON values.

These members are the claims represented by the JWT.

The member names within the JWT Claims Set are referred to as Claim Names.

The corresponding values are referred to as Claim Values.

JWT Claims Set

A JSON object that contains the claims conveyed by the JWT.

JSON Web Token (JWT)

A string representing a set of claims as a JSON object that is encoded in a JWS or JWE, enabling the claims to be digitally signed or MACed and/or encrypted.

Claim

A piece of information asserted about a subject.

A claim is represented as a name/value pair consisting of a Claim Name and a

Claim Value.

Claim Name

The name portion of a claim representation.

A Claim Name is always a string.

Claim Value

The value portion of a claim representation.

A Claim Value can be any JSON value.

example of 3 claims:

"sub": "1234567890"

"name": "John Doe"

"iat": 1516239022

JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based data structures.

JSON Web Encryption (JWE) represents encrypted content using JSON-based data structures.

JSON Web Algorithms (JWA)

JSON Object Signing and Encryption (JOSE)

Registered Claim Names

----------------------

"iss" (Issuer) Claim

"sub" (Subject) Claim

"aud" (Audience) Claim

"exp" (Expiration Time) Claim

"nbf" (Not Before) Claim

"iat" (Issued At) Claim

"jti" (JWT ID) Claim

Public Claim Names

------------------

Claim Names can be defined at will by those using JWTs.

However, in order to prevent collisions, any new Claim Name should either be registered in the IANA "JSON Web Token Claims" registry established.

Public Name: a value that contains a Collision-Resistant Name.

Private Claim Names

-------------------

A producer and consumer of a JWT MAY agree to use Claim Names that are Private Names.

Private Names: names that are not Registered Claim Names or Public Claim Names.

Unlike Public Claim Names, Private Claim Names are subject to collision and should be used with caution.

JOSE Header(JSON Object Signing and Encryption (JOSE)) Header

-------------------------------------------------------------

For a JWT object, the members of the JSON object represented by the JOSE Header describe the cryptographic operations applied to the JWT and optionally, additional properties of the JWT.

Depending upon whether the JWT is a JWS or JWE, the corresponding rules for the JOSE

Header values apply.

"typ" (Type) Header Parameter

"cty" (Content Type) Header Parameter

example:

"alg": "HS256",

"typ": "JWT"

<https://tools.ietf.org/html/rfc7519#section-5>

**Logic to create JWT token:**

String key="raju";

String jwts = Jwts.*builder*()

.setIssuer("Msoft")

.setSubject("john")

.setAudience("cns")

.setIssuedAt(**new** Date(System.*currentTimeMillis*()))

.setExpiration(**new** Date(System.*currentTimeMillis*()+60000))// will be expired after 1 min

.signWith(SignatureAlgorithm.***HS256***, key)

.compact();

System.***out***.println(jwts);

**Decoding header and payload**

System.***out***.println("decoding token:");

StringTokenizer st = **new** StringTokenizer(jwts,".");

String h = **new** String(Base64.*getDecoder*().decode(st.nextToken()));

System.***out***.println("header:"+h);

String p = **new** String(Base64.*getDecoder*().decode(st.nextToken()));

System.***out***.println("payload:"+p);

Parsing the token:

**try** {

//get parser instance

JwtParser jws = Jwts.*parser*();

jws.setSigningKey(key);

Jwt jwt= jws.parse(newJwts);

// we can safely trust the JWT

System.***out***.println("valid token");

// take header and body

String header = jwt.getHeader().toString();

String body = jwt.getBody().toString(); // means pay load

System.***out***.println(header);

System.***out***.println(body);

}

**catch** (JwtException ex) {

//we \*cannot\* use the JWT as intended by its creator

System.***out***.println("Error!!:"+ex.getMessage());

}