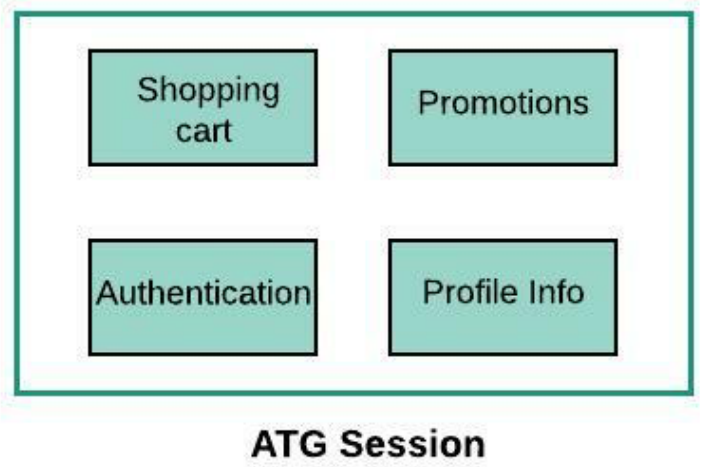
ATG Sessions & NGC Statelessness

What is a ATG Session

ATG session is an object maintained by ATG in memory for the duration of 15 minutes(*minimum*) whenever a request (**by guest or a job or a system**) hits ATG. Once a session starts , every request need to pass the ID of this session (**JSESSIONID**) on all subsequent request to let ATG know who the guest is (anonymous or logged in). A session also works as authentication mechanism in ATG as session holds guest authentication & authorization (**Secure Resource Access**) in ATG framework .

Session typically includes profile information , current order information , last order placed , Promotions of a guest etc.



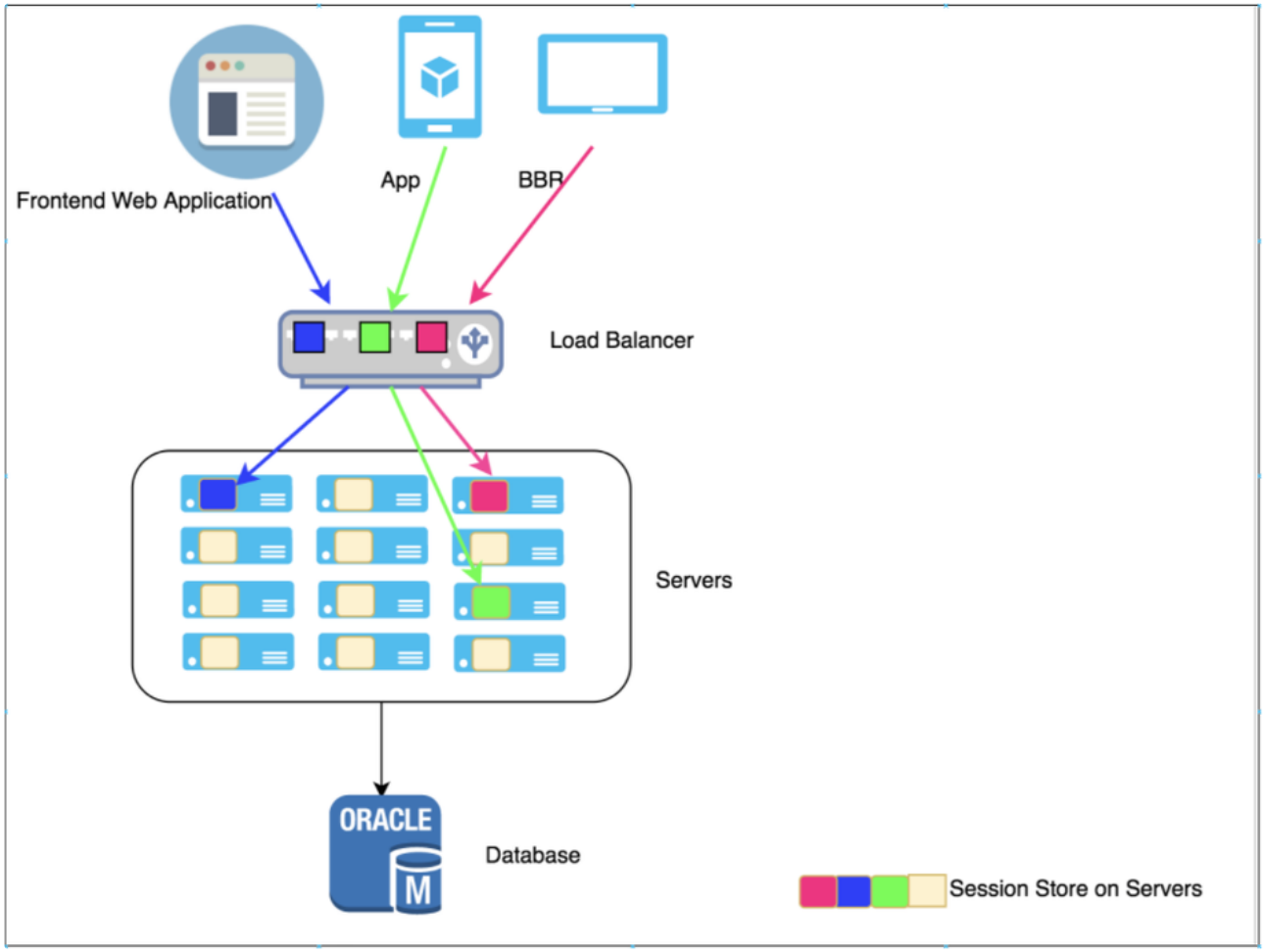
ATG platform has its own Session wrapper over Http Session provided by Application server (Weblogic). This is must to configure for an ATG application to work .

Quick snapshot about ATG session:

- ATG keeps session object **in memory** of a particular JVM

- Sessions are not available globally outside a JVM to other servers.
- ATG performance comes with keeping sessions in memory of a JVM
- **JSESSIONID** is identifier for a session .
- **JSESSIONID** need to be passed to each ATG request by browser, app, graphQL, BBR or any other system to maintain guest experience intact throughout the guest journey.

How ATG Session works



ORACLE Commerce

admin/

Component Browser

Search Context

Service /atg/dynamo/servlet/sessiontracking/GenericSessionManager/

Class atg.multisite.session.MultisiteComponentSessionManager

View Service Configuration

Service Info

The SessionManager used in JavaEE and Servlet containers

Session Manager Admin

Enter a comma delimited list of keys that should be used to look up session values (case is important; e.g. "Name,Age"). If no list is provided then, as a default, the id, creation and last accessed time will be displayed for each session.

View

Properties

Name	Value	Type
absoluteName	/atg/dynamo/servlet/sessiontracking/GenericSessionManager	java.lang.String
adminServlet	instanceOf atg.servlet.sessiontracking.SessionViewerAdminServlet	javax.servlet.Servlet
adminServletOutputStreamEncoding	UTF-8	java.lang.String
adminServletUseServletOutputStream	false	boolean
attributeNames	instanceOf atg.core.util.IteratorEnumeration	java.util.Enumeration
attributeSerializationBug	true	boolean
backingUpSessions	false	boolean
errorURL	null	java.lang.String
inheritsEffectiveScope	false	boolean
initParameterNames	null	java.util.Enumeration
licenseChecklist	null	[Ljava.lang.String;
logListenerCount	2	int
logListeners	atg.nucleus.logging.LogListener [2]	[Latg.nucleus.logging.LogListener;
loggingDebug	false	boolean
loggingError	true	boolean
loggingInfo	true	boolean
loggingTrace	false	boolean
loggingWarning	true	boolean
majorVersion	2	int
managedScopeName	session	java.lang.String
maxSessionCount	-1	int
mimeTyper	null	atg.servlet.MimeTyper
minorVersion	2	int
name	GenericSessionManager	java.lang.String
nameContext	/atg/dynamo/servlet/sessiontracking	atg.naming.NameContext
nameContextBindingListeners	atg.naming.NameContextBindingListener [1]	[Latg.naming.NameContextBindingListener;
nameContextManagerDebug	false	boolean
nameContextPreBindingListeners	atg.naming.NameContextPreBindingListener [2]	[Latg.naming.NameContextPreBindingListener;
nameForJSessionId	JSESSIONID	java.lang.String
noSessionMimeTypes	java.lang.String [2]	[Ljava.lang.String;
nucleus	/	atg.nucleus.Nucleus
optimizedForConcurrency	true	boolean
pingParentWebAppMillis	3000	long
root	/	atg.naming.NameContext
running	true	boolean
serverInfo	null	java.lang.String
serviceConfiguration	PropertyConfiguration	atg.nucleus.Configuration
serviceInfo	The SessionManager used in JavaEE and Servlet containers	java.lang.String
servletNames	instanceOf atg.core.util.IteratorEnumeration	java.util.Enumeration
servlets	instanceOf java.util.Vector$1	java.util.Enumeration
sessionBackupPropertyList	java.lang.String [8]	[Ljava.lang.String;
sessionCookieName	JSESSIONID	java.lang.String
sessionCount	384	int
sessionSaverTools	/atg/dynamo/servlet/sessiontracking/SessionSaverTools	atg.servlet.sessionsaver.SessionSaverTools
sessionURLName	jsessionid	java.lang.String
singleSessionIdPerUser	false	boolean
siteManager	/atg/multisite/SiteManager	atg.multisite.SiteManager
startServiceComplete	true	boolean
useSessionTrackingCookie	false	boolean

Note :

- With OKTA Login going live this year, ATG treats **OKTA token** as master & auto-login the guest if token is valid. ATG session always have 15 minutes time-out irrespective of OKTA time-out. Thats where front end (web, app, graphQL) has to perform a re-login with ATG whenever there is a mismatch between **OKTA & ATG authentication state.**
- **Common misconception** is as OKTA providing login, why ATG has to maintain session ? ATG session is fundamental to ATG architecture irrespective of login happens outside ATG or using ATG login module.

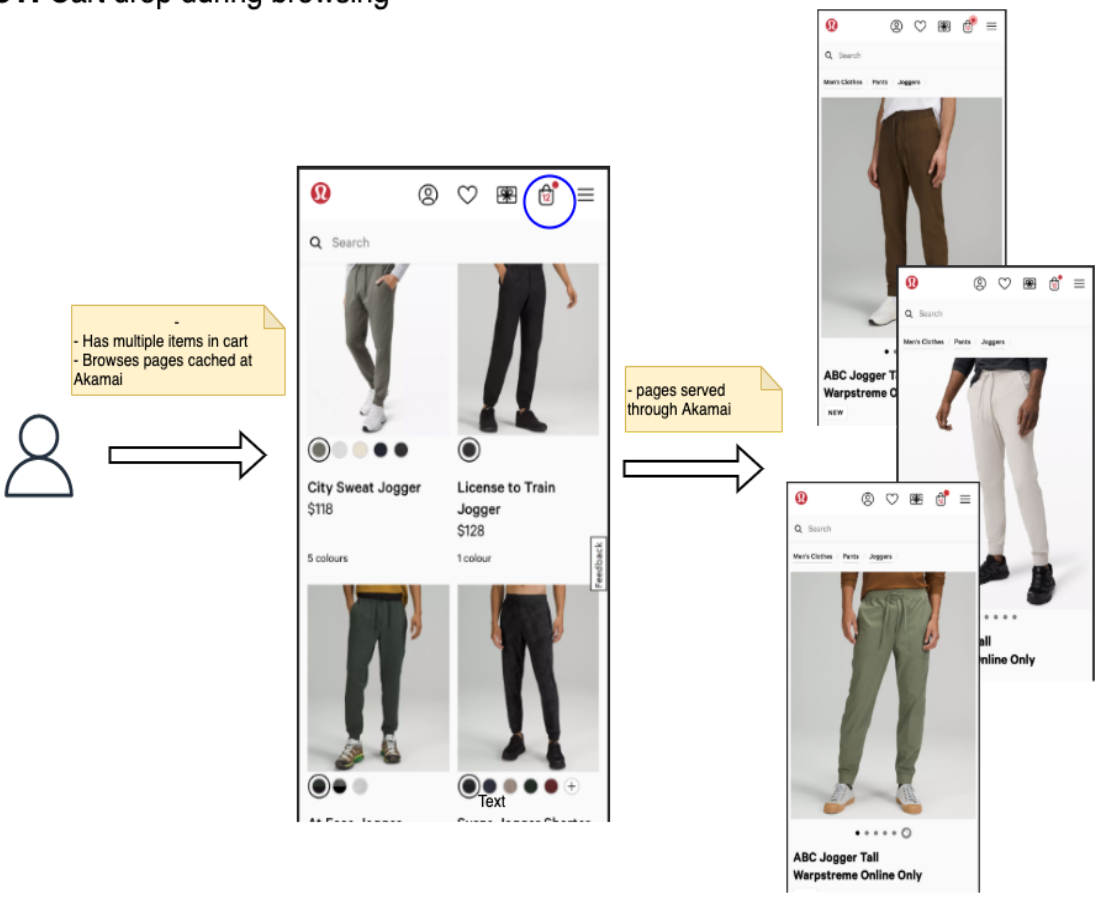
Issues with ATG session based architecture

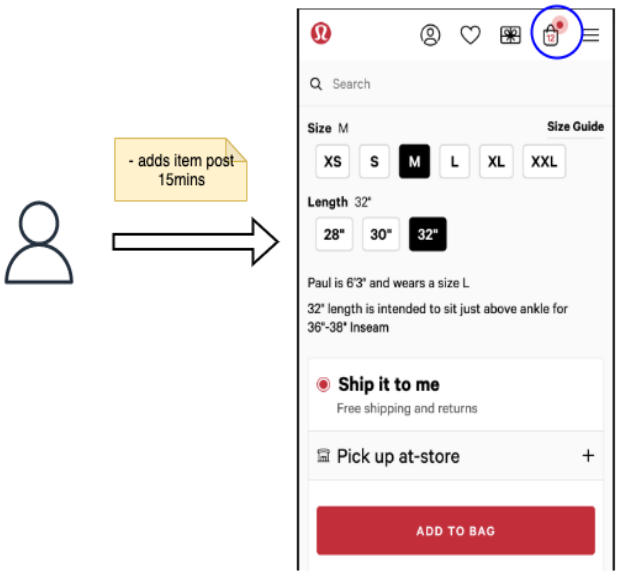
Cart Drop Scenarios

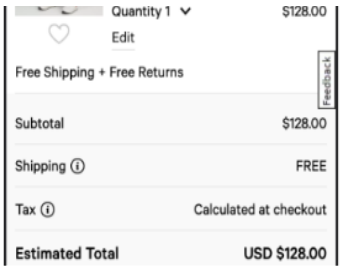
Guest journey 1

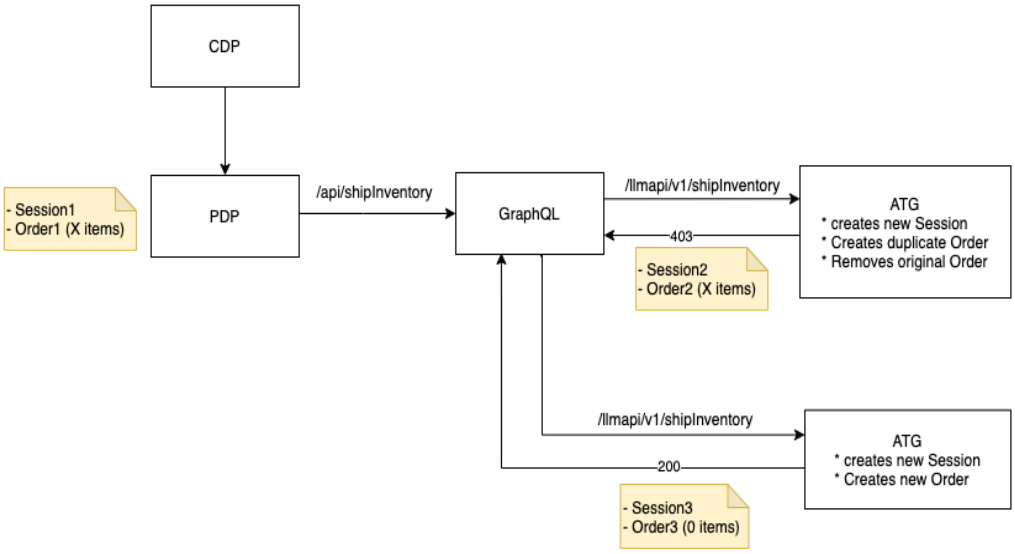
- Logged -in Guest keeps browsing CDP, PDP , Story, Search pages after initial interaction with ATG for more than 15 minutes
- ATG expires the session because of 15 minutes inactivity
- JSESSIONID for guest session expires
- Guest see cached X items in My cart icon in top header.
- Guest clicks add to cart , view cart - sees empty cart or a warning on shopping cart to re-login.
- On re-login merge cart function triggers and empties entire cart in certain conditions (Code issues in merge function).

Scenario1: Cart drop during browsing

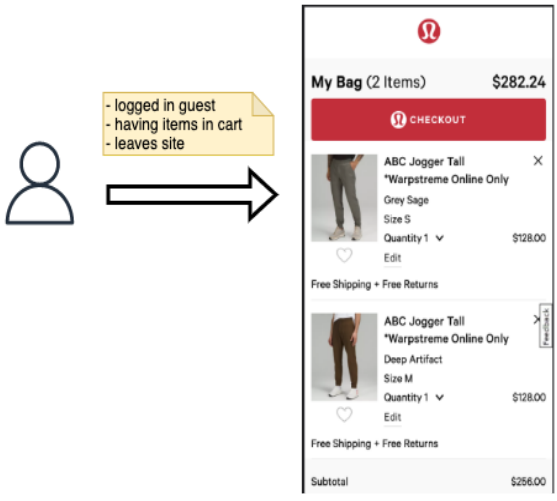


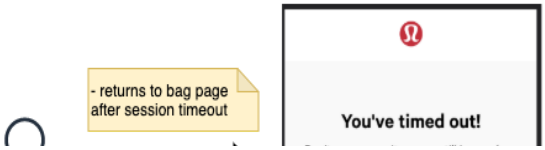


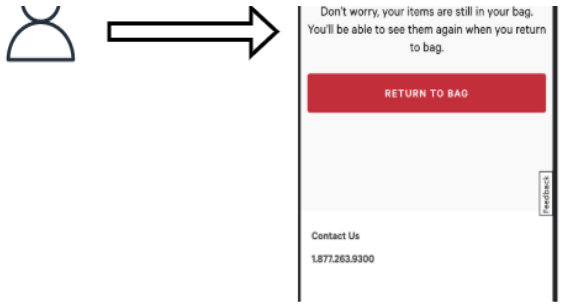


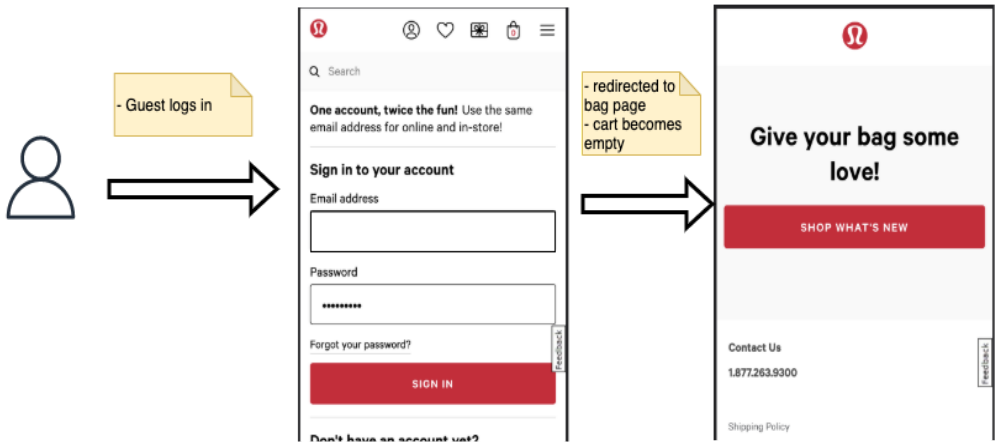


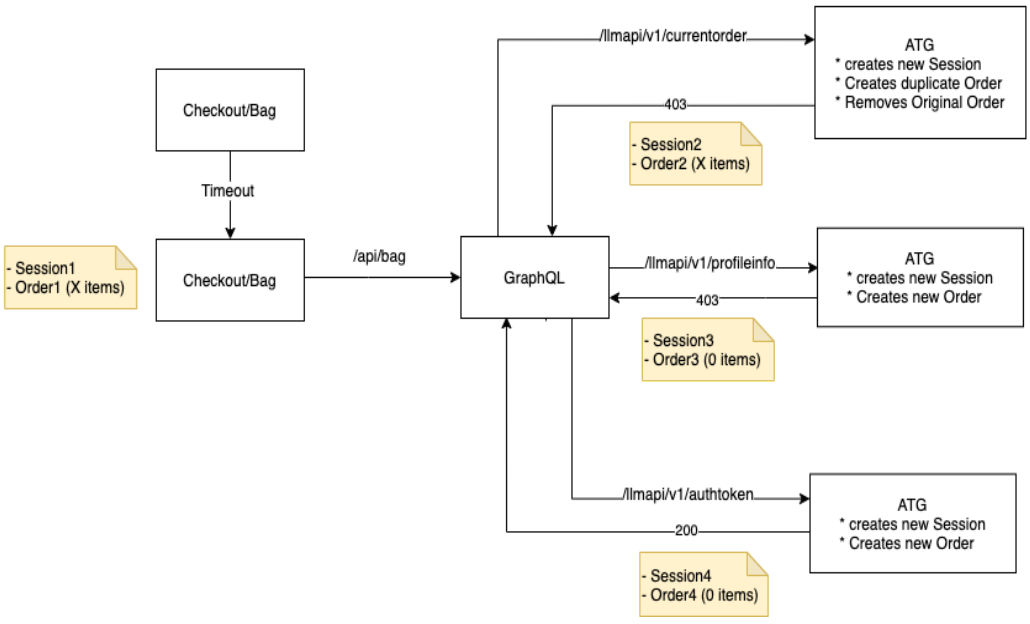
Scenario2: Cart drop after sessiontim eout











Guest journey 2

- Guest (anonymous or logged-in) is in lower funnel (shipping, billing, payment, final review)
- Guest request switches to a different JVM because of JSESSIONID cookie corruption via load balancer issue, GraphQL issues etc.
- Guest will either lose entire checkout information & will go back to shopping cart page.

- This is typical session drop issue & creates inconvenience for guests.

ATG System Issues because of session based architecture

- As Sessions hold memory , a particular JVM memory can go higher , resulting in system instability & experience instability for all guests whose requests are being served by that particular JVM.
- Longer release times as it takes time to drain all sessions on each JVM in a origin , which makes release time longer as well as guests experience issues where sessions get lost.
- Limits scalability of ATG architecture as instances can't be auto-scaled to handle increased traffic without impacting guest experience.
- In a distributed architecture , whenever a tier (Front end, GraphQL, ATG) corrupts JSESSIONID cookie , session is not retrievable at all & guests have to start afresh. ATG team has developed a lot of complex code to handle these situations which is now hard to understand & debug.

NGC Stateless architecture

What is statelessness ?

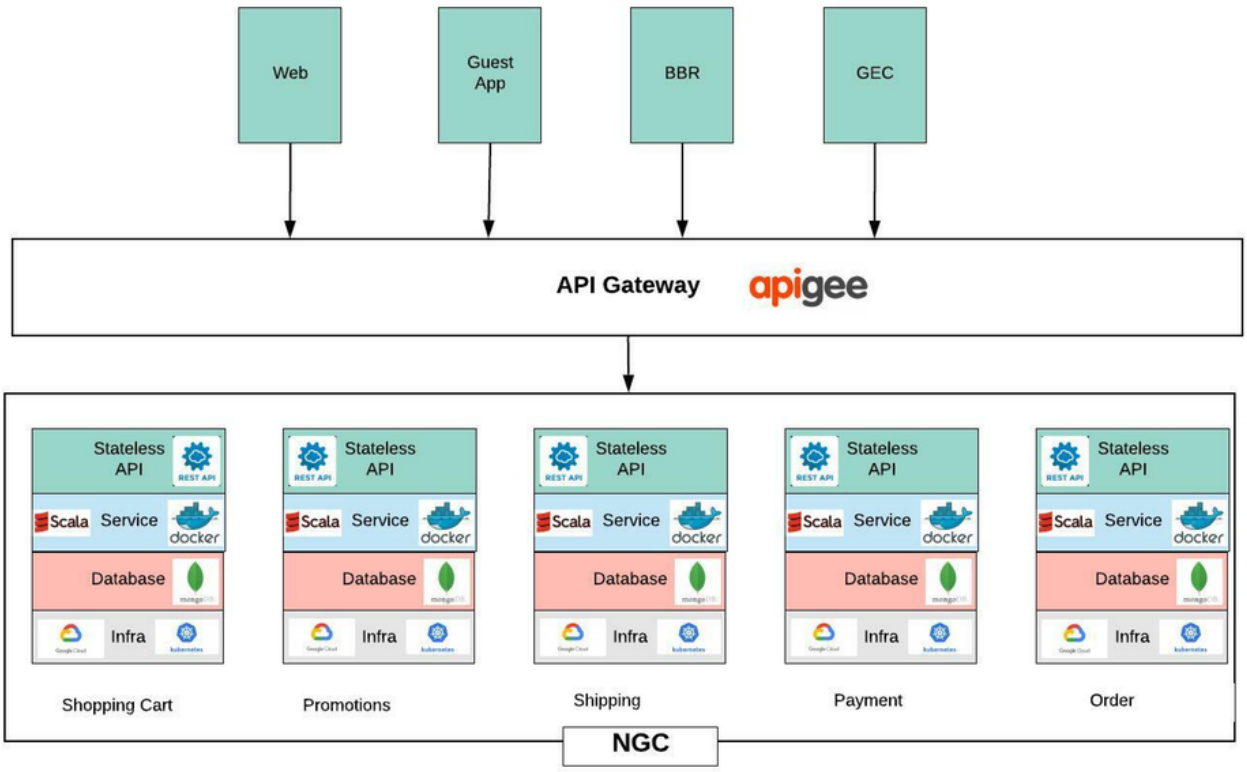
Service statelessness is a principle whereby a service that provides a function is decoupled from the state required to perform that function. This is done by having the state data passed to the service as part of the function request or by externalizing the state to a horizontal scalable external storage layer.

Key Differences b/w Stateless and Stateful architecture

	Stateless Architecture (NGC)	Stateful Architecture (ATG)
1	**No Session** : The server processes requests based only on information relayed with each request and doesn't rely on information from earlier requests – this means that the server doesn't need to hold onto state information between requests.	**Session Based** : The server processes requests based on the information relayed with each request and **information stored from earlier requests** – this means that the server must access and hold onto state information generated during the processing of the earlier request.
2	**No Sticky Session :** Different requests can be processed by different servers	**Sticky Session** :**The same server** must be used to process all requests linked to the same state information, or the state information needs to be shared with all servers that need it.
3	**Highly Scalable** :Any service instance can retrieve an application state necessary to execute a behavior from elsewhere enables **resiliency, elasticity**, and **availability**.	**Limited Scalability :** All requests in a session need to reach same server which makes it **less resilient and elastic**.

NGC architecture

- Ephemeral containers
- auto-scaling at each tier
- True Restful APIs
- Cloud Native
- 12 factor principles



How NGC architecture solves issues coming through ATG session based architecture

External Storage of state

- **No Sessions** in memory (Sessionless architecture)
- Shopping Cart is **always persisted** in external MongoDB database (statelessness)
- With speed of MongoDB database , latest document based architecture, CQRS pattern - there is no overhead in writing & reading data for **each request (~50ms**) from an external database like MongoDB
- MongoDB being a NO-SQL database **horizontally scalable** , this allows service layer (**containers**) to be auto-scalable as traffic increases without impacting guest experience
- As there is no state being maintained in container memory, **traffic can go to any container for each request** & still get the same response & performance.
- Enables **quick & frequent releases** (multiple times a day) as we don't need to wait to drain sessions . We can quickly release any feature with percentage rollout.

Tokens instead of Session IDs

- Shopping Cart is **always bound to Omni-id** if a guest is logged-in. Complements OKTA's token based architecture.
- NGC APIs authentication is based on Oauth2.0 tokens from OKTA (Logged-in guests) or CommerceTools (anonymous guests).
- Tokens are validated with each request through a token service(anonymous) or using cached public keys(OKTA)
- Token provides secure access to resources based upon claims in token

NGC API Design

- APIs are truly Restful which clearly states APIs should be stateless
- APIs are idempotent for GET, PUT operations
- APIs will be made idempotent wherever needed as per business requirements

- APIs access is based on Oauth2.0 based tokens
- APIs will provide correct status based on resource state for each request