Microsoft Machine Learning Server

The Operationalization Engine for your Advanced Analytics

Microsoft

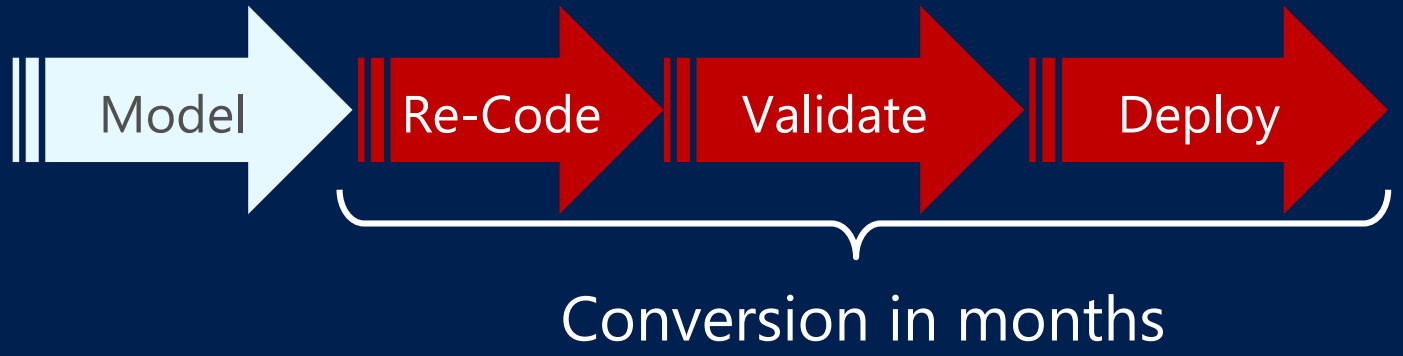
Predictive Analytics Process @ high level

Prepare: Assemble, cleanse, profile and transform diverse data relevant to the subject.

Model: Use of statistical and machine learning algorithms to build classifiers and predictions

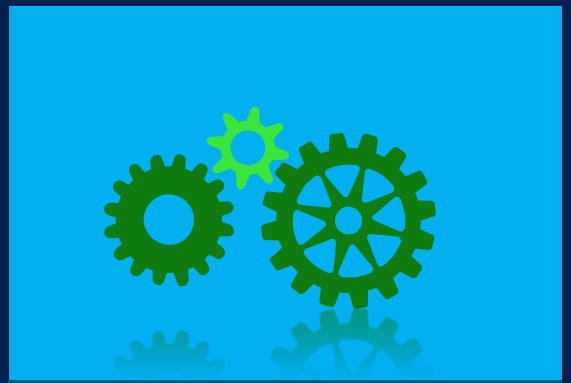
Operationalize: Apply predictions and visualizations to support business applications

Challenge: Long Deployment Lifecycle



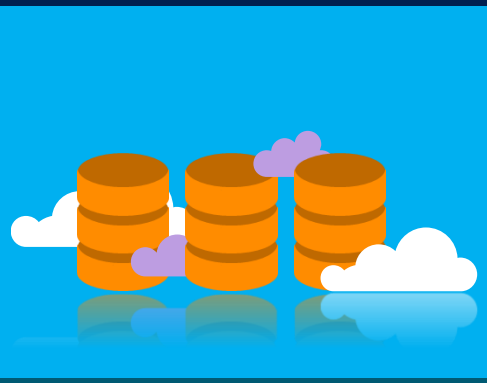
- Results:
  - Slow innovation rates
  - Stale models
  - Errors
  - Extended testing & validation cycles

Microsoft Machine Learning Server
The Operationalization Engine of your Advanced Analytics



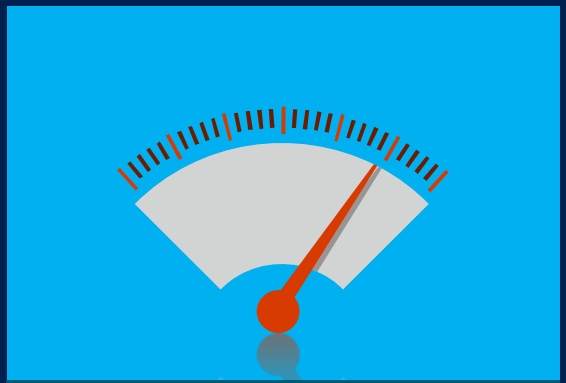
Instant Deployment

- Turn R/Python analytics → Web services in one line of code;
- Swagger-based REST APIs, easy to consume, with any programming languages!



Deploy to Anywhere

- Deploying web service server to any platform: Windows, SQL, Linux/Hadoop
- On-prem or in cloud



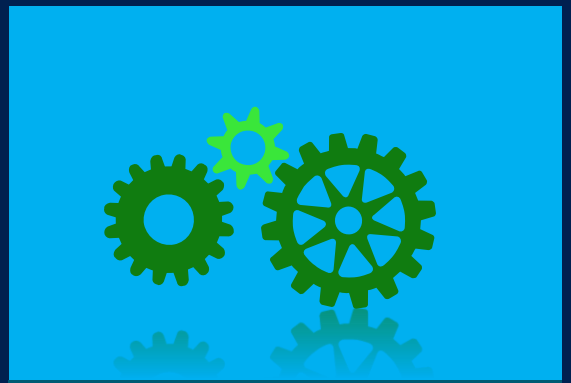
Fast and Scalable

- Fast scoring, real time and batch
- Scaling to a grid for powerful computing with load balancing
- Diagnostic and capacity evaluation tools

Secure and Reliable

- Enterprise authentication: AD/LDAP or AAD
- Secure connection: HTTPS with SSL/TLS 1.2
- Enterprise grade high availability
- Role Based Access Control

Microsoft R Server
The Operationalization Engine of your Advanced Analytics



Instant Deployment

- Turn R/Python analytics → Web Service in one line of code;
- Swagger-based REST APIs, easy to consume, with any programming languages!

Deploy to Anywhere

- Deploying Web Service server to any platform: Windows / SQL / Linux/Hadoop
- On Prem or in Cloud

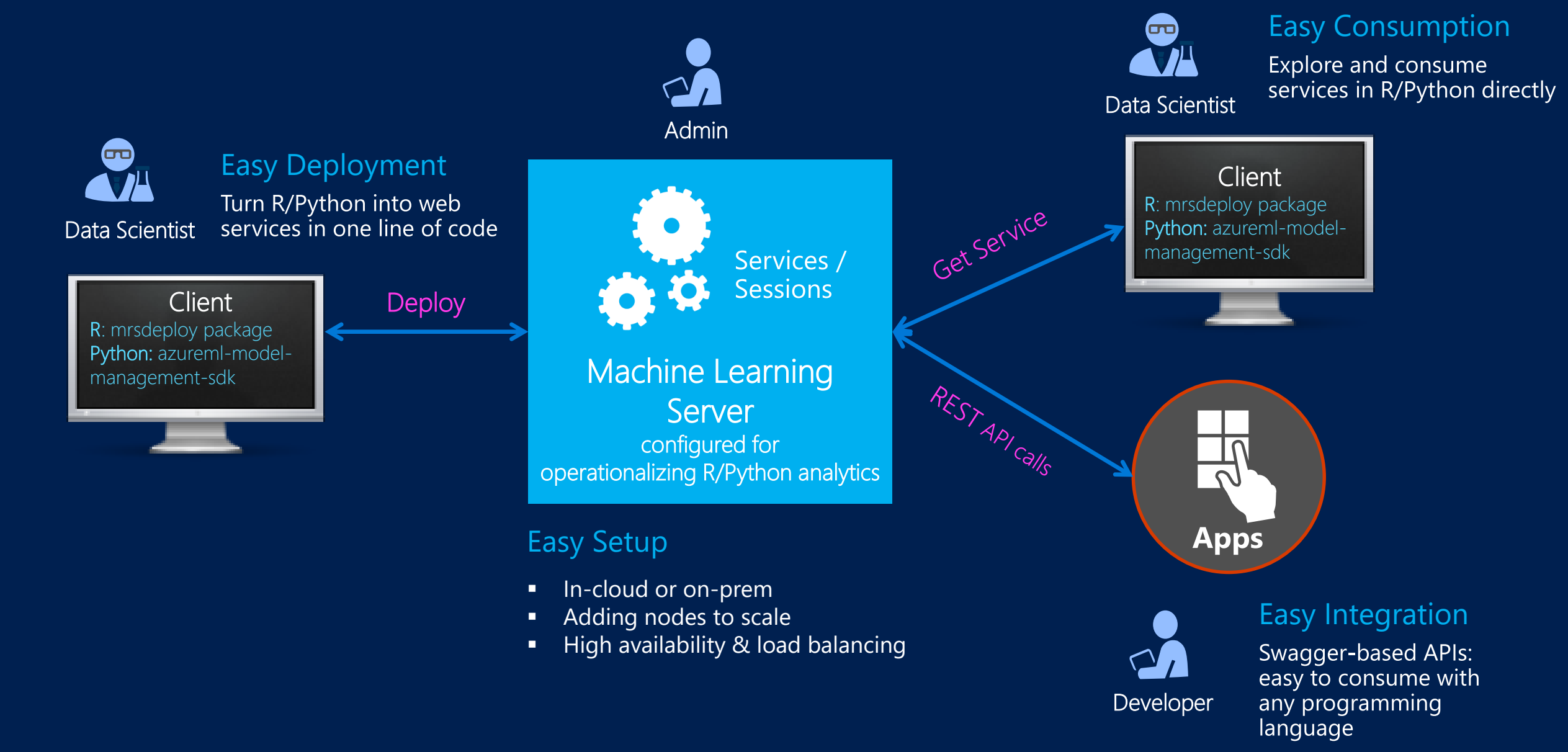
Fast and Scalable

- Fast scoring, real time and batch
- Scaling to a grid for powerful computing with load balancing
- Diagnostic and capacity evaluation tools

Secure and Reliable

- Enterprise authentication: LDAP / AD/ AAD
- Secure connection: HTTPS with SSL.TSL1.2
- Enterprise grade High Availability

Rapid Deployment for Intelligent Production Apps



Instant Deployment

Turn R/Python into Web Services in one line of code

Build the model first

```r
# --- Build the model first  -----------------------

model <- glm(formula = am ~ hp + wt,
    data = mtcars,
    family = binomial)

# --- Wrap into a prediction function  -------------

manualTransmission <- function(hp, wt) {
    newdata <- data.frame(hp = hp, wt = wt)
    predict(model, newdata, type = "response")
}
```

Deploy as a web service instantly

```r
# --- Access R Server ------------------------------

remoteLoginAAD(
  "https://deployr-dogfood.contoso.com",
  authuri = "https://login.contoso.net",
  tenantid = "contoso.com",
  clientid = "3955bff3-2ec2-4975-9068-2812345a3b6f",
  resource = "b3b96d00-1c06-4b9d-a94f-1234571822b0",
  session = FALSE
)

# --- Deploy as web service ----------------------------

api <- publishService(
   serviceName,
   code = manualTransmission,
   model = "transmission.RData",
   inputs = list(hp = "numeric", wt = "numeric"),
   outputs = list(answer = "numeric"),
   v = "v1.0.0"
)

# --- Consume the service right away in R! -------------
result <- api$manualTransmission(120, 2.8)
```

Integration with Apps

Swagger based APIs, easy to consume, with any programming language

Data Scientist

Developer

Developer

Generate Swagger Docs for Web Services

Run Swagger tools to generate code

Write a few code to consume the service

```
# Run the following code

swagger <- api$swagger()

cat(swagger, file = "swagger.json",
append = FALSE)
```

Popular Swagger Tools:
AutoRest or Code Generator

*AutoRest.exe -CodeGenerator CSharp -Modeler Swagger -Input **swagger.json** -Namespace Mynamespace*

```
using System;
using MyNamespace;
using MyNamespace.Models;

namespace TransmissionApiExample
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var api = new Transmission(new Uri("https://rservertest.com"));
            var accessToken = "{{YOUR_JWT_TOEKN}}";

            var headers = client.HttpClient.DefaultRequestHeaders;
            headers.Remove("Authorization");
            headers.Add("Authorization", $"Bearer {accessToken}");

            InputParameters inputs = new InputParameters() { hp = 120, wt = 2.8 };
            var serviceResult = api.Manual.TransmissionAsync(inputs).Result;

            Console.Out.WriteLine(serviceResult.OutputParameters);
        }
    }
}
```

Boost up scoring performance with real-time web services

Realtime web service vs. Standard web service

Impact by algorithm complexity

Algos	Real time (ms)	Standard (ms)
RxLogit (model size 2K)	3.5	39.2
RxNeuralNet (model size 8K)	2.5	122.0

Impact by model size

Model Size	Real time (ms)	Standard (ms)
2 MB (RxLogisticRegression)	5.0	9215.7
43 MB (RxLogisticRegression)	5.4	20255.6*

* Server: Standard_D3_v2 (4 CPU core, 14GB RAM)

As low as <10ms latency

10X or even 100X faster than standard web services

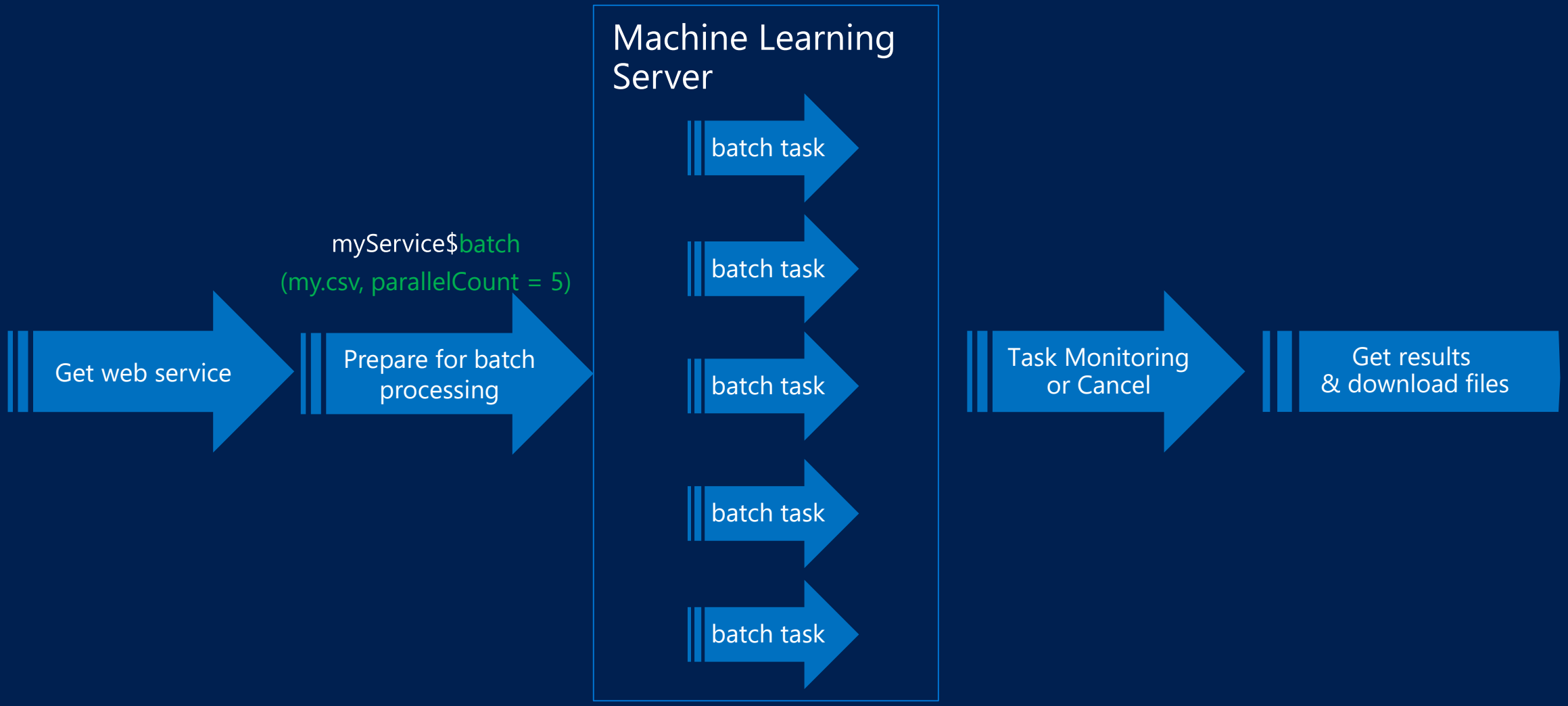
Why Fast?

- Score with native code, not R (R interpreter is slow)
- Cache the model in memory (load only once)

* Only models built with RevoScaleR algos and MicrosoftML algos can be deployed as Realtime web services and hosted in Windows Server. Additional platform support will come from future releases.

Asynchronously batch processing

For large input data sets and long running computations



Easy Consumption of services with R/Python

Enabling exciting new scenarios for data scientists

Enable **Model Management** capabilities

- A Predictive Web Service = "Model" + "Prediction Script"
- R Server hosts all those services → Central Repo of Models
- Each service has a version tag → Model Version Control
- All versions are active → Model Roll Back (to any version)
- A service can be accessed by any authorized users →
  - Model reuse
  - Model validation and monitoring by QA team

After service is published, I can test if the service works as expected right away

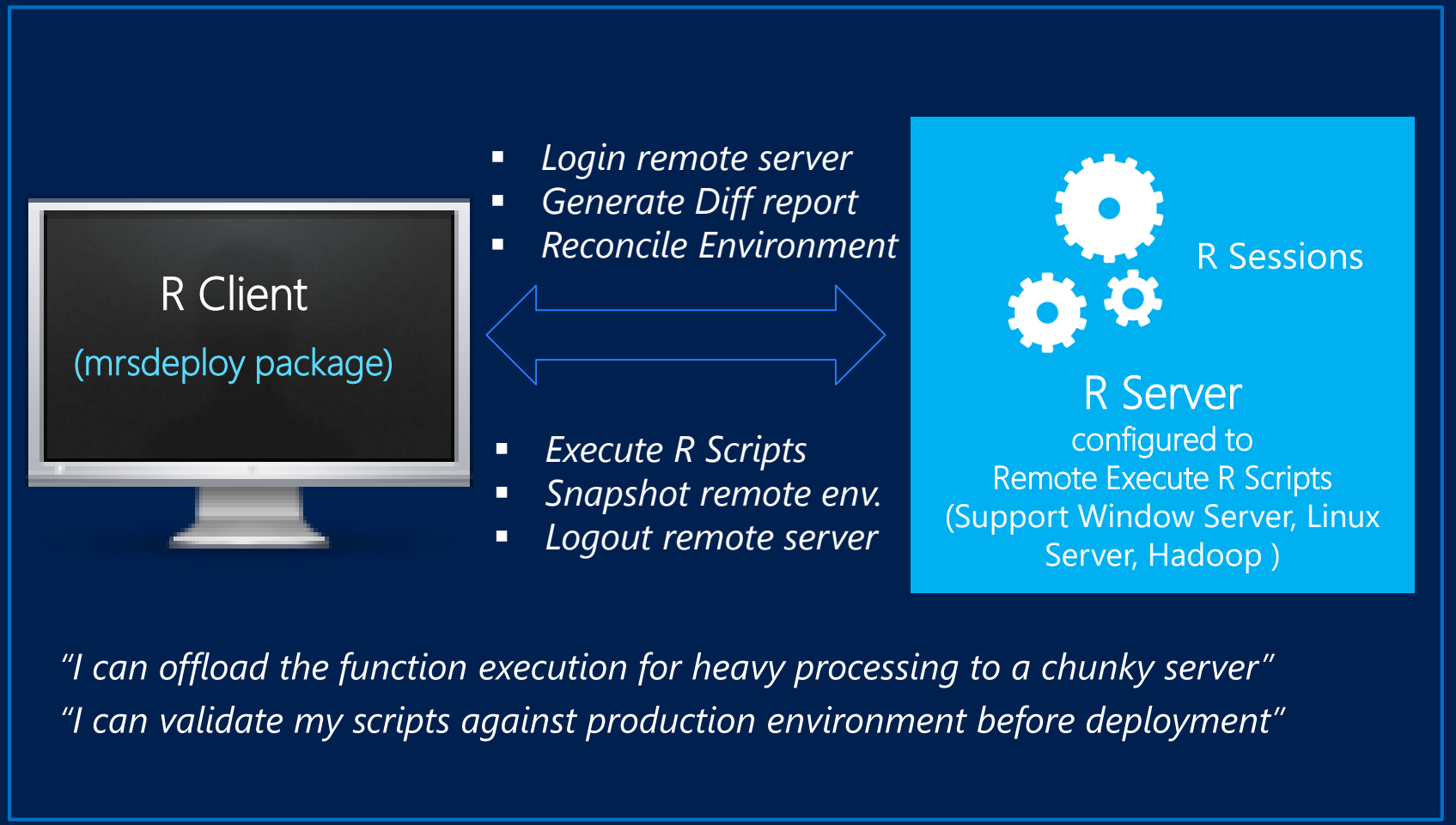
Share / Reuse code / functions

- Not just models, a data scientist can share any functional code as a service.
- Other data scientists can explore in the repository to re-use those functions.

Remote Execute R scripts

Configure R Server to host remote R sessions

- Built-in remote execute functions in R Client/R Server
- Generate Diff report to reconcile local and remote
- Execute .R script or interactive R commands
- Results come back to local
- Generate working snapshots for resume and reuse
- IDE agnostic



Please note: remote execute doesn't support Python yet.

Remote Execution Cheat Sheet

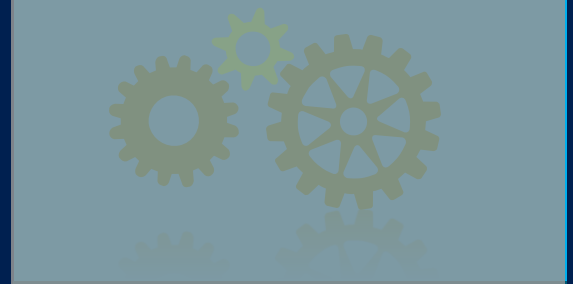
Remote Connection	
remoteLogin	Remote login to the R Server with AD or admin credentials
remoteLoginAAD	Remote login to R Server server using Azure AD
remoteLogout	Logout of the remote session on the DeployR Server.

Remote Execution	
remoteExecute	Remote execution of either R code or an R script
remoteScript	Wrapper function for remote script execution
diffLocalRemote	Generate a 'diff' report between local and remote
pause	Pause remote connection and back to local
resume	Return the user to the 'REMOTE >' command prompt

Snapshot Functions	
createSnapshot	Create a snapshot of the remote session (workspace and working directory)
loadSnapshot	Load a snapshot from the server into the remote session (workspace and working directory)
listSnapshots	Get a list of snapshots for the current user
downloadSnapshot	Download a snapshot from the server
deleteSnapshot	Delete a snapshot from the server

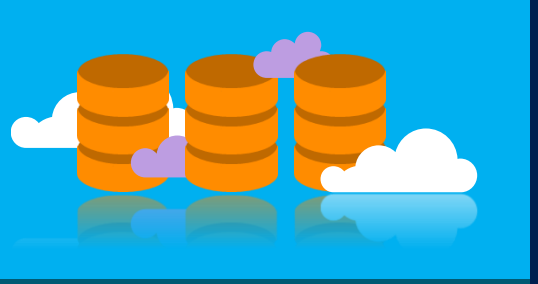
Remote Objects Management	
listRemoteFiles	Get a list of files in the working directory of the remote session
deleteRemoteFile	Delete a file from the working directory of the remote R session
getRemoteFile	Copy a file from the working directory of the remote R session
putLocalFile	Copy a file from the local machine to the working directory of the remote R session
getRemoteObject	Get an object from the remote R session
putLocalObject	Put an object from the local R session and load it into the remote R session
getRemoteWorkspace	Take all objects from the remote R session and load them into the local R session
putLocalWorkspace	Take all objects from the local R session and load them into the remote R session

Microsoft R Server
The Operationalization Engine of your Advanced Analytics



Instant Deployment

- Turn R analytics → Web Service in one line of code;
- Swagger-based REST APIs, easy to consume, with any programming languages, including R!



Deploy to Anywhere

- Deploying Web Service server to any platform: Windows / SQL / Linux/Hadoop
- On Prem or in Cloud

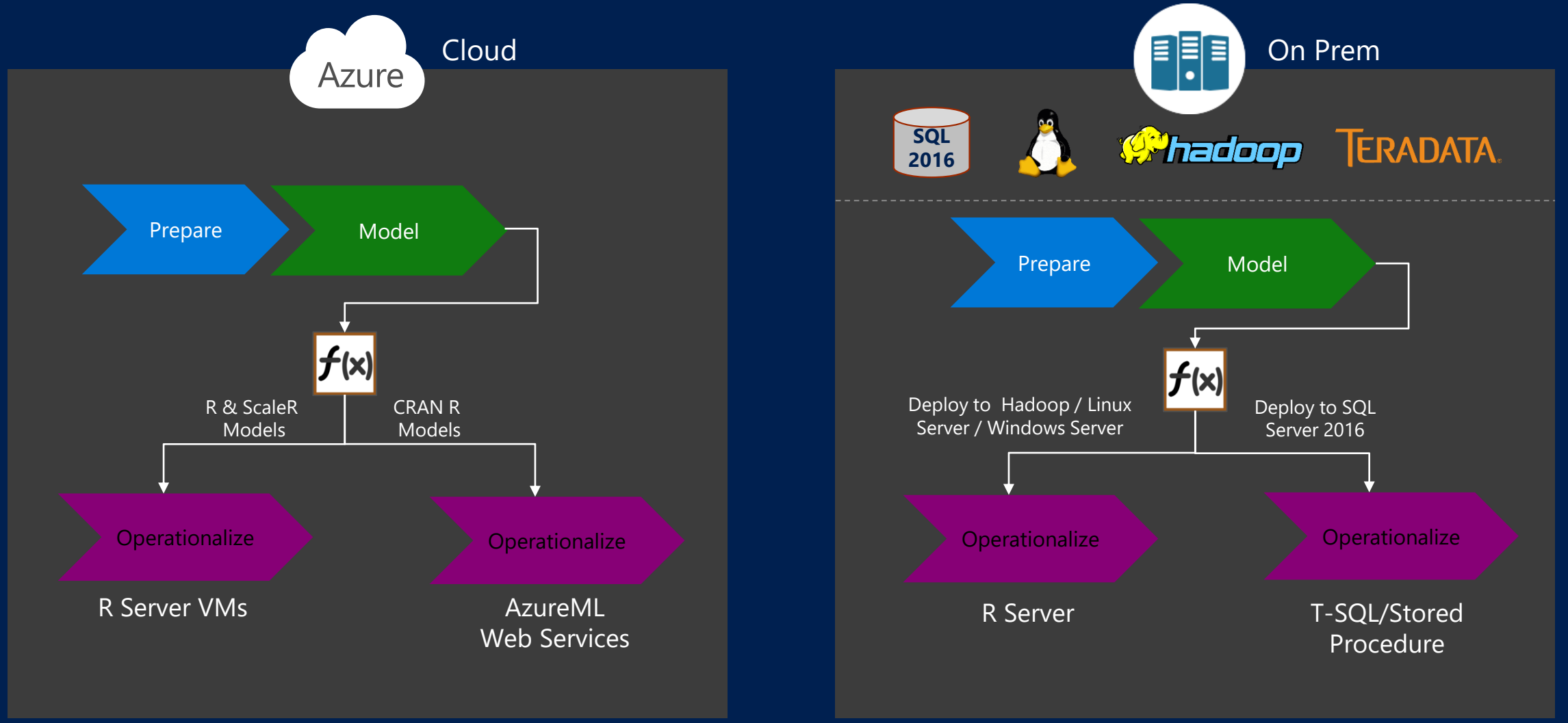
Fast and Scalable

- Fast scoring, real time and batch
- Scaling to a grid for powerful computing with load balancing
- Diagnostic and capacity evaluation tools

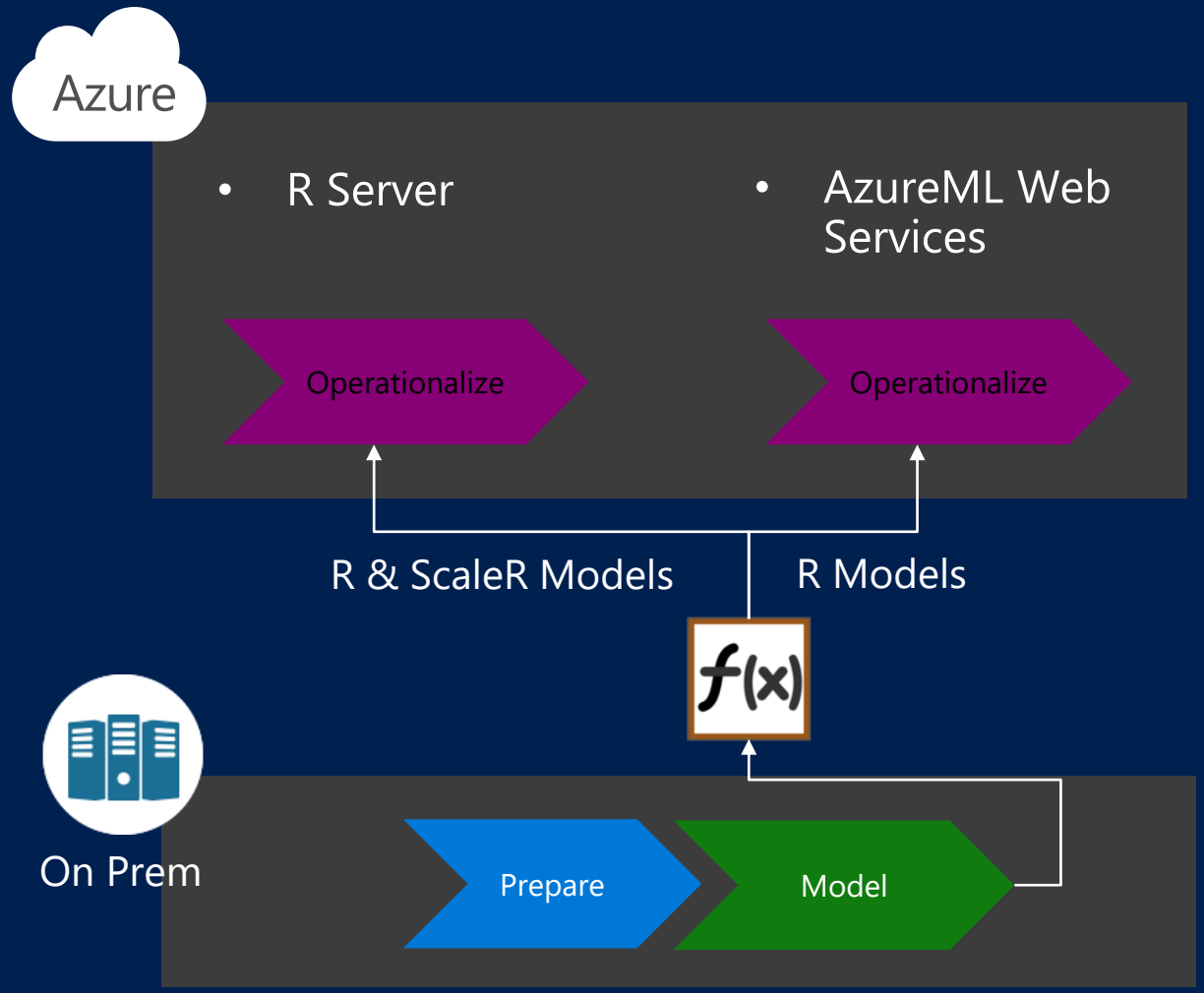
Secure and Reliable

- Enterprise authentication: LDAP / AD/ AAD
- Secure connection: HTTPS with SSL.TSL1.2
- Enterprise grade High Availability

Deploy to Anywhere: On Prem or In Cloud



Deploying Hybrids: Data Lake on prem, Predictive Action in Cloud



Hybrid Cloud Deployment:

- Data Lake on prem
- Deploy Models to Azure
- Expose Services via Azure/R Server

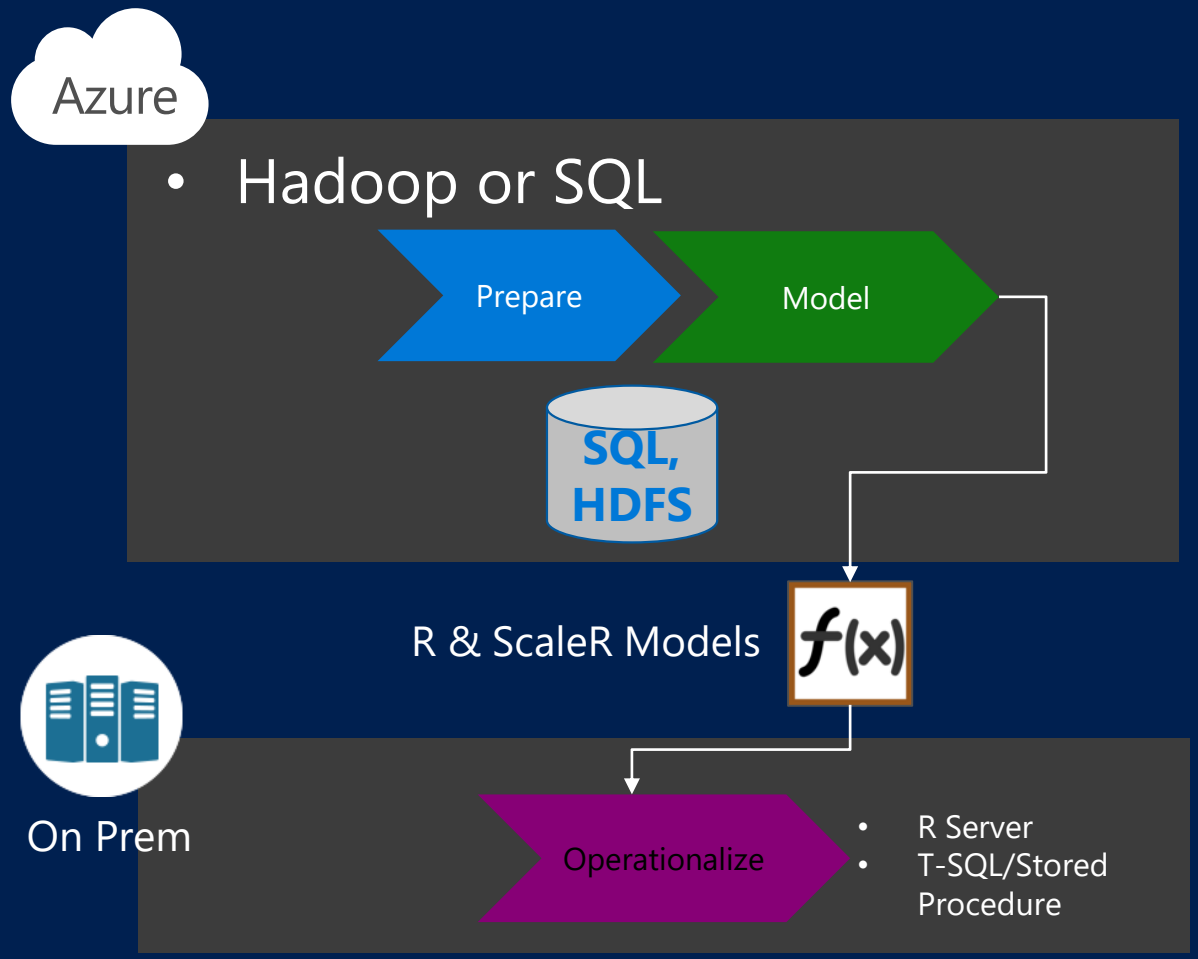
Advantages:

- Score "Born In The Cloud" Data
- Deploy Globally via Cloud Services
- Secure Historic Data Locally

Uses:

- Auto Insurance Quote for Connected Car – realtime-data from dongle in cloud, historical data on-prem, modeling on-prem; Scoring for auto insurance quote, in cloud

Deploying Hybrids: Cloud Modeling; On-Prem Prediction



Model in Azure:

- Capture in Data Lake
- Explore & Transform in R/Python

Deploy On-Prem.

- Scoring and BI Visualizations
- Expose Web Services

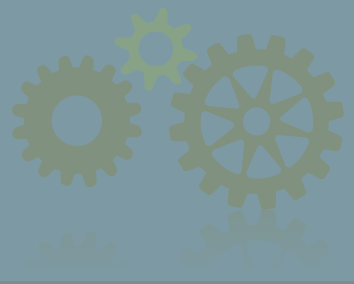
Advantages:

- Cloud Economics & Scale for Big Data
- SQL Server Stability, Privacy for Deploy

Examples:

- Manufacturing Process Optimization – Oil rigs
- Point-of-Sale Anomaly (fraud) Detection

Microsoft R Server
The Operationalization Engine of your Advanced Analytics



Instant Deployment

- Turn R analytics → Web Service in one line of code;
- Swagger-based REST APIs, easy to consume, with any programming languages, including R!

Deploy to Anywhere

- Deploying Web Service server to any platform: Windows / SQL / Linux/Hadoop
- On Prem or in Cloud

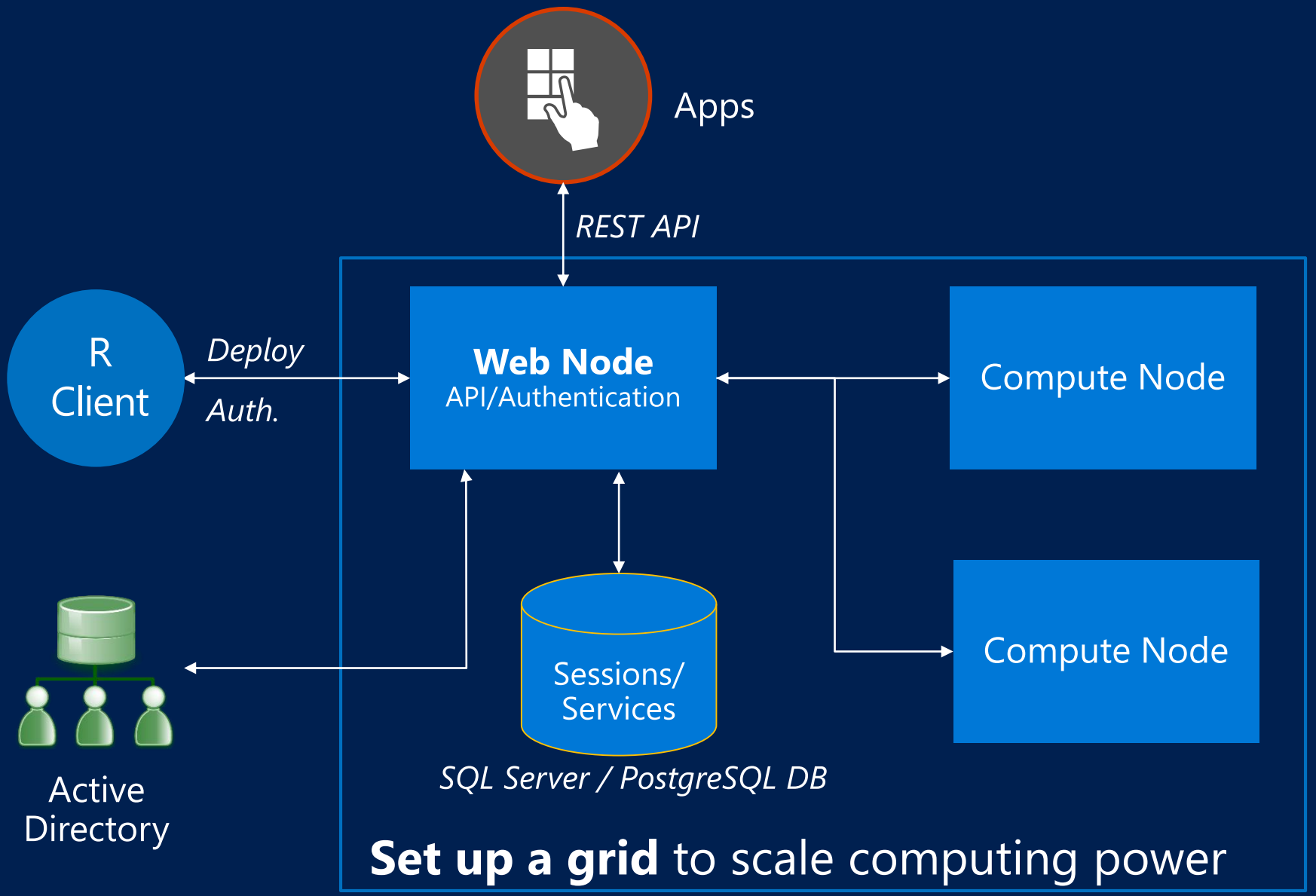
Fast and Scalable

- Fast scoring, real time and batch
- Scaling to a grid for powerful computing with load balancing
- Diagnostic and capacity evaluation tools

Secure and Reliable

- Enterprise authentication: LDAP / AD/ AAD
- Secure connection: HTTPS with SSL.TSL1.2
- Enterprise grade High Availability

Scale up for more powerful computing



- Easily scale up a single server to a grid to handle more concurrent requests
- Load balancing cross compute nodes
- A shared pool of warmed up R shells to improve scoring performance.

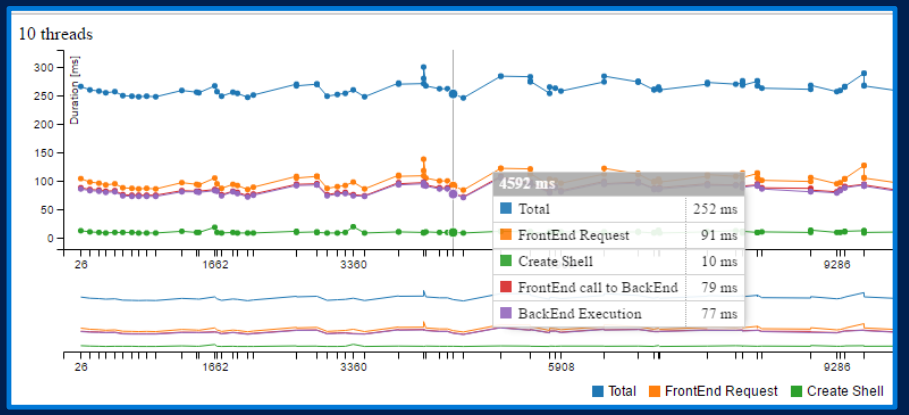
Diagnostic and Evaluation Tools

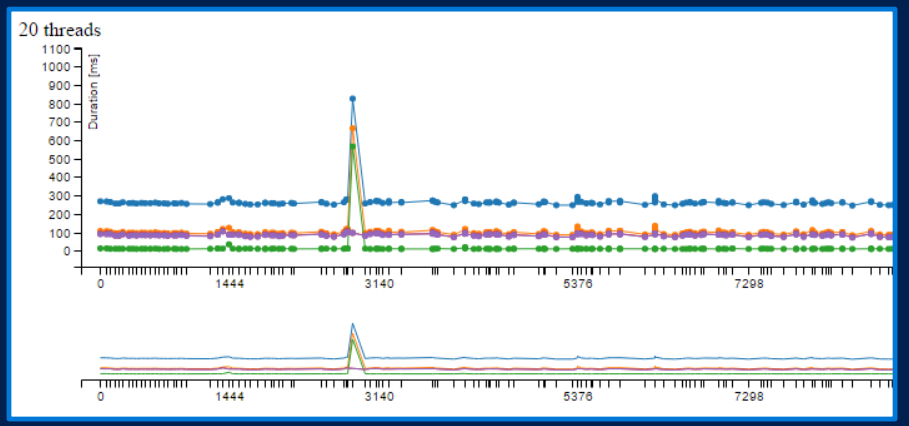
Diagnostic Tool

- Health check node configuration
- Get system status
- Trace R code execution
- Trace service execution

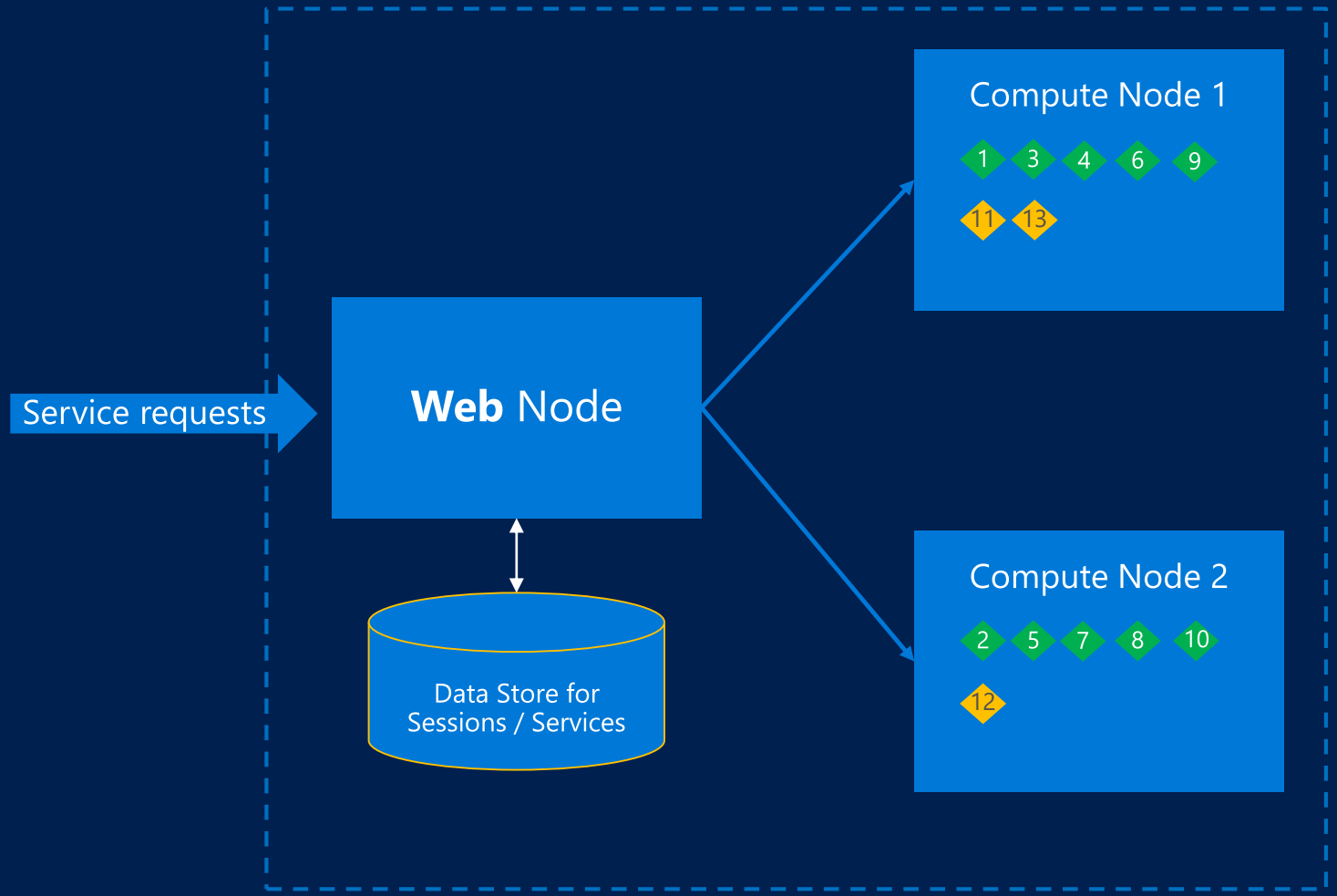
Evaluation Tool

- Evaluate grid capacity
- Simulate traffic per service
- Configure with # of concurrent threads or latency thresholds





Load Balancing among Compute Nodes



1 Service / Session running in an R Shell

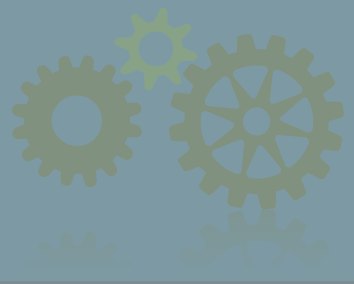
13 New R Shell created to host additional services/sessions

- Shell: each service or session is hosting in a Shell with R/Python interpreter up and running.
- Shell Pool: each node maintains a pool of Shells (warm pool). These Shells can run simultaneously by leveraging multiple cores of the node.
- Initial Pool Size is 5 (configurable): When a node starts, it will initialize 5 Shells.
- Incoming services will be distributed to compute nodes evenly.
- When concurrent services exceed current pool size, pool can grow automatically by adding new shells to pool. This does take time (~1s).
- Max Pool Size is 80 (configurable): when max pool size reached, a new Shell will be created and discarded right after usage.

Sizing the Grid

- No universal formula to estimate the grid size.
- Key factors to impact the grid size:
  - Compute power of compute nodes: CPU, # of cores and memory.
  - The throughput of services (i.e., how many incoming service requests per seconds).
  - The expected latency for request-response time.
  - The complexity of the R/Python Code: CPU consumption and memory consumption.
- Using diagnostic tool to test against a one-box setup is the best practice.

Microsoft R Server
The Operationalization Engine of your Advanced Analytics



Instant Deployment

- Turn R analytics → Web Service in one line of code;
- Swagger-based REST APIs, easy to consume, with any programming languages, including R!

Deploy to Anywhere

- Deploying Web Service server to any platform: Windows / SQL / Linux/Hadoop
- On Prem or in Cloud

Fast and Scalable

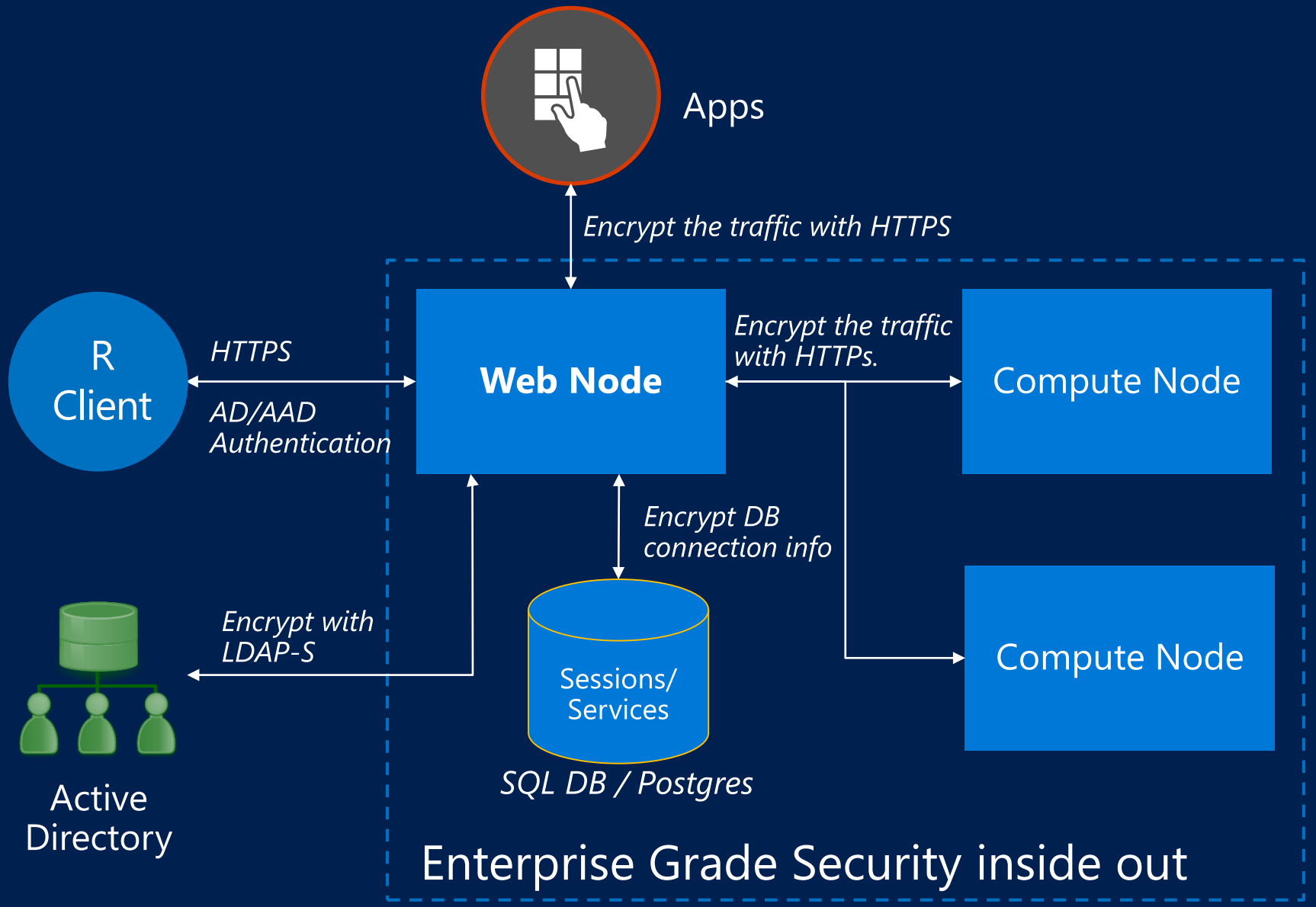
- Fast scoring, real time and batch
- Scaling to a grid for powerful computing with load balancing
- Diagnostic and capacity evaluation tools

Secure and Reliable

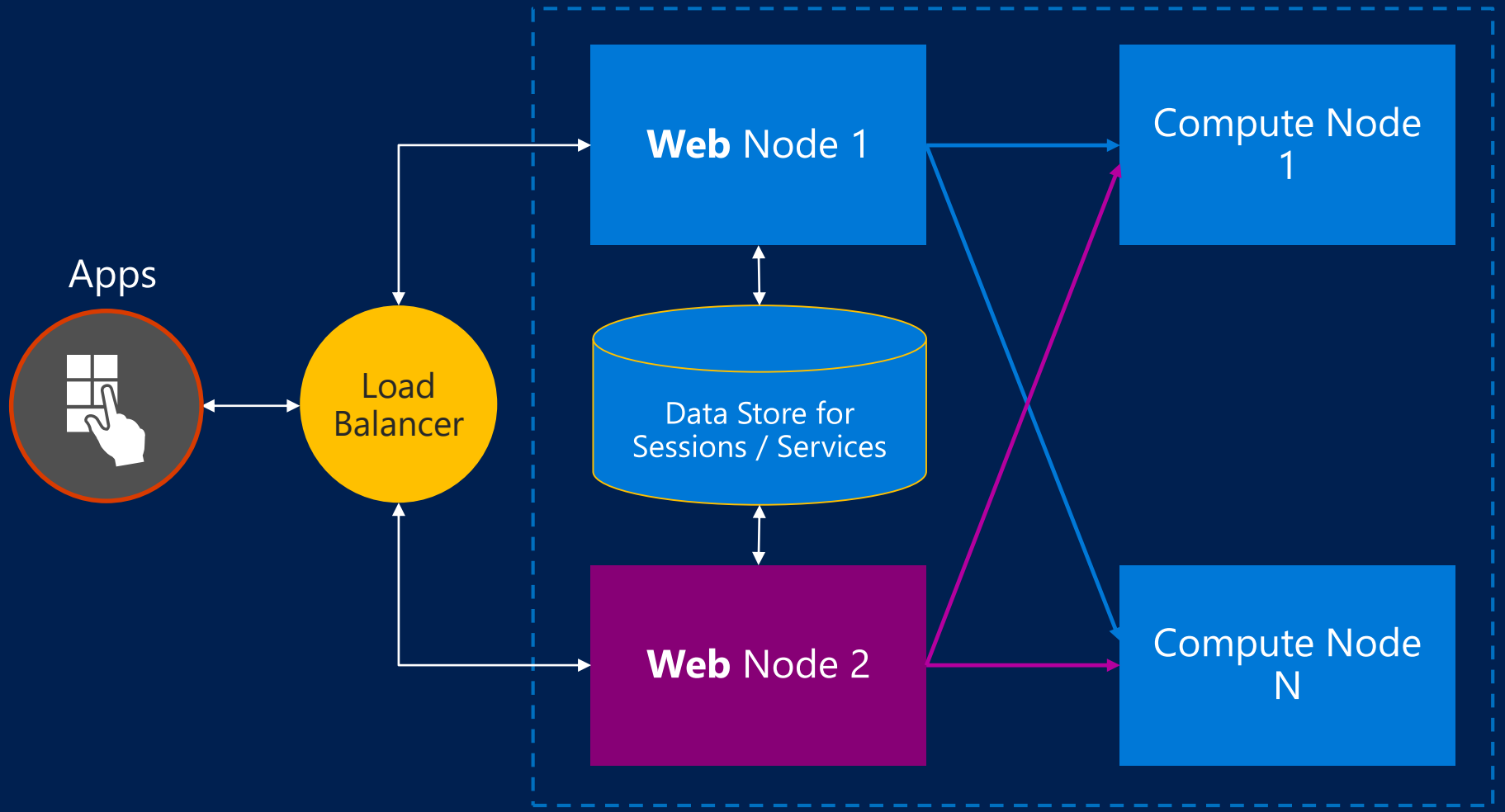
- Enterprise authentication: LDAP / AD/ AAD
- Secure connection: HTTPS with SSL.TSL1.2
- Enterprise grade High Availability
- Role Based Access Control

Enterprise Grade Security

- Seamless integration with authentication solution: LDAP/AD/AAD
- Secure connection: HTTPS encrypted by TLS 1.2/SSL
- Compliance with Microsoft Security Development Lifecycle
- Role Based Access Control



High Availability (disaster recovery)



- Server level HA: Introduce multiple Web Nodes for Active-Active backup / recovery, via load balancer
- Data Store HA: leverage Enterprise grade DB, SQL Server and Postgres' HA capabilities

Role Based Access Control

Enable admins to further control who can publish, update and delete web services

Role	Can do with web services	Cannot do with web services
Owner	• Publish/Update/Delete **any** service • List all services • Consume any service	None
Contributor	• Publish/Update/Delete **their** services • List all services • Consume any service	Update / Delete service published by someone else
Reader	• List all services • Consume any service	Publish/Update/Delete any service
Others*	• Nothing (cannot even login to Server)	Anything

*: introduced since Machine Learning Server 9.2

Backup Slides

Comparison with previous solution

	Microsoft R Server 8.0.5: DeployR	Machine Learning Server 9.0 Operationalization
Installation	A separate installer from R Server	ML Server includes all deployment capabilities; **Greatly improved the installation and configuration experience.**
Deployment (turn R analytics into web services)	Involve multiple steps, and by default uploading R analytics to Repo DB is the first step.	**Publish your R/Python analytics** as web services directly from your R console **with one line of code.**
Consumption of web services	Integrate with app via Client library; RBroker Framework.	**Easy to integrate with Apps using Swagger** based REST API; Enable many exciting scenarios by **consuming services directly in R/Python!**
Enable Remote Execution	Customers have to use DeployR APIs to build their own way of remote execution	**Built-in remote execution functions** in 'mrsdeploy' package in R Client/R Server.
Architecture	Apache Tomcat	**ASP .Net Core. Cross-platform, better support,** endorsed by Microsoft.
Authentication	Basic/AD/LDAP/PAM authentication	AD/LDAP/**Azure AD** auth.
High Availability	Doesn't support Active-Active recovery, unless clone another grid.	Support **Active-Active recovery** via multiple web nodes.
Web UI	Login/Admin Console/Repository Manager/API Explorer/Event Console	• **Coming soon in future releases** • Totally new design, ease of use.
APIs	~100 DeployR APIs	**Simplified APIs.** ~40 raw APIs. Not compatible with 8.0.x.

Supported Platforms

- Machine Learning Server operationalization built on top of .Net Core which doesn't support some of the OS versions.
- In product documentation, will state which platforms can be configured to operationalize R/Python.
- Will fill the gaps in the future releases.

Product	Platforms	Modeling	Operationalization
ML Server for Windows	Windows Server	2012 - 2016	Same as modeling
ML Server for Linux	Red Hat Enterprise Linux	6.X and 7.X	7.x
ML Server for Linux	SUSE Enterprise	SLES 11	will support in future release
ML Server for Linux	Ubuntu	14.04 LTS, 16.04 LTS	Same as modeling
ML Server for Linux	CentOS	6.X and 7.X	7.x
ML Server for Hadoop	Red Hat and SUSE Enterprise	RHEL 6.x and 7.x, SUSE SLES11	RHEL 7.x

Web Service Functions Cheat Sheet for R

Function	Description
publishService	Publish a predictive function as a Web Service
deleteService	Delete a Web Service
getService	Get a Web Service
ListServices	List the different published web services
serviceOption	Retrieve, set, and list the different service options
updateService	Updates a Web Service

Publish a Web Service

The `publish_service` function publishes a new web service.

Arguments

- `name` - (Required) Defines the name of the service
- `code` - (Required) Defines the R code that will be ran. The provided `code` value can either be:
  i. A filepath to an R script `code = "/path/to/R/script.R"`
  ii. A block of R code as a character string `code = "result <- x + y"`
  iii. A function handle:

```
code = function(hp, wt) {
  newdata <- data.frame(hp = hp, wt = wt)
  predict(model, newdata, type = 'response')
}
```

- `model` - (Optional) A filepath to a binary object `.RData` file or a filepath to an R Script
- `inputs` - (Optional) A `List` which defines the web service input schema
- `outputs` - (Optional) A `List` which defines the web service output schema
- `v` - (Optional) Defines a unique web service version
- `alias` - (Optional) The predication RPC function used to consume the service
- `descr` - (Optional) The description of the web service.

Response

An *Api* instance as an R6

What's new in Machine Learning Server 9.2

New Features in Machine Learning Server 9.2

- Deploy Python code as web services
- Extend real time web services support from Windows to Linux, and to Python.
- Add a new role in RBAC to prohibit that role from accessing the server.
- Manage compute nodes from Admin Util

Deploy Python code as web services

- azureml-model-management-sdk
- Functions on part with those of R's
- Fluent API to easily deploy Python models

```
# Authentication
from azureml.deploy import DeployClient
from azureml.deploy.server import MLServer

HOST = '{{https://YOUR_HOST_ENDPOINT}}'
context = ('{{YOUR_USERNAME}}',
'{{YOUR_PASSWORD}}')
client = DeployClient(HOST, use=MLServer,
auth=context)
```

```
# Deploy as web service
service = client.service('myService')\
        .version('v1.0')\
        .description('cars model')\
        .code_fn(manualTransmission, init)\
        .inputs(hp=float, wt=float)\
        {{...}}
        .deploy()
```

What's new in R Server 9.1

Boost up scoring performance with real-time web services

Realtime web service vs. Standard web service

Impact by algorithm complexity

Algos	Real time (ms)	Standard (ms)
RxLogit (model size 2K)	3.5	39.2
RxNeuralNet (model size 8K)	2.5	122.0

Impact by model size

Model Size	Real time (ms)	Standard (ms)
2 MB (RxLogisticRegression)	5.0	9215.7
43 MB (RxLogisticRegression)	5.4	20255.6*

* Server: Standard_D3_v2 (4 CPU core, 14GB RAM)

As low as <10ms latency

10X or even 100X faster than standard web services

Why Fast?

- Score with native code, not R (R interpreter is slow)
- Cache the model in memory (load only once)

* Only models built with RevoScaleR algos and MicrosoftML algos can be deployed as Realtime web services and hosted in Windows Server. Additional platform support will come from future releases.

Role Based Access Control

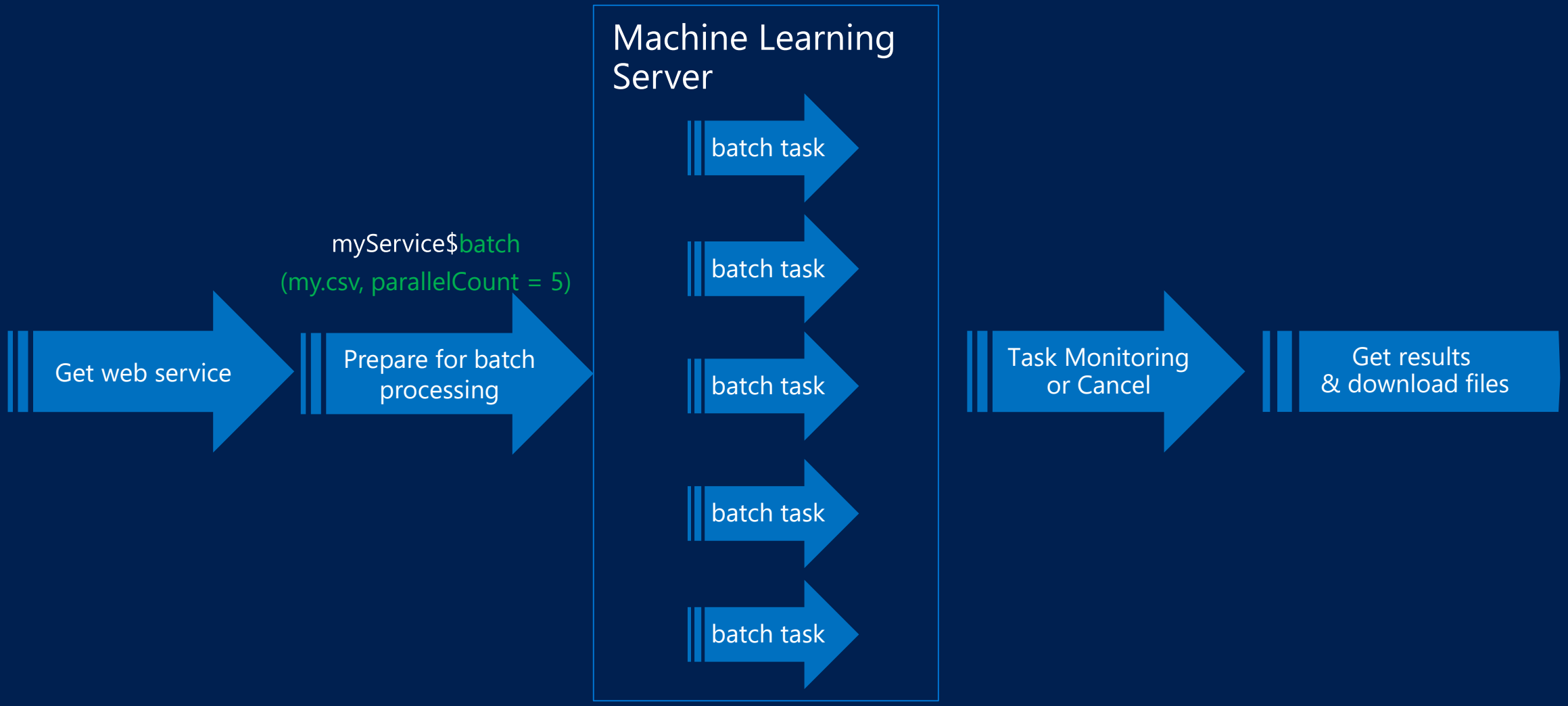
Enable admins to further control who can publish, update and delete web services

Role	Can do with web services	Cannot do with web services
Owner	• Publish/Update/Delete **any** service • List all services • Consume any service	None
Contributor	• Publish/Update/Delete **their** services • List all services • Consume any service	Update / Delete service published by someone else
Reader	• List all services • Consume any service	Publish/Update/Delete any service
Others*	• Nothing (cannot even login to Server)	Anything

*: introduced since Machine Learning Server 9.2

Asynchronously batch processing

For large input data sets and long running computations



Asynchronous remote execution

Run scripts in background mode, without having to wait for the job to complete

```
#switch to the local R session
REMOTE > pause()

#execute an R script remotely
> remoteScript("C:/myScript.R")

#execute that script again in another window asynchronously
> remoteScript("C:/myScript.R", async = TRUE)
```

Auto scaling of operationalization grid on Azure

Offer an ARM Template / Script to:

- Spin up a set of R Server VMs in Azure
- Configure them as a grid for operationalization
- Scale it up and down based on CPU / Memory usage.

R Server vs Shiny

R Server

Integrating R with LOB apps
with any programming language

Data Scientist + Developer

Target use case:

Integrate Real time quote in current insurance system.

Shiny

Sharing analysis result
via interactive web apps

Data Scientist

Target use case:

Interactive dashboard with R visualization

Please refer to this blog for detail comparison