Problem 4- Ml Cloud Platforms.

S. No	Parameter	IBM Cloud Platform	Google Cloud Platform	Microsoft	Amazon
1.	DL Frameworks	IBM Watson Studio, Watson ML- Pytorch, Tensorflow, Caffe, Keras,	Deep Learning VM on compute engine- Pytorch, Keras+Tensorflow, Chainer, MXNet, RAPIDS Xgboost, CNTK	Azure ML- Support for Caffe, Caffe2, Chainer(5.2), CUDA, Horovod, Keras, CNTK, MXNet, Pytorch, Tensorflow, Theano, Tensorflow Servig	Amazon DLAMI- pytorch, tensorflow, MxNet, with CUDA instances Amazon SageMaker- pytorch, tensorflow, MxNet, with CUDA instances
2.	Compute Units	BareMetal Severs, Virtual servers, Application Servers	• General Purpuse computing (n1) • webserver (n2) • HPC (c2) • Memory Optimised • CPU Optimised All machines are customizable in terms of Memory and CPU	• General Purpose (DC, Av2, Dv2 and Dv3) • Compute optimised Fsv2, F series) • Memory Optimised(M, G, EV3) • Storage Optimised(LS) • GPU Based • High Perf Computing (H series)	
3.	Types of supported storage	HDD and SSD	HDD and SSD Persistent storage	HDD and SSD	HDD and SSD
4.	Explicit support for RL?	Models can be deployed on IBM Cloud, no explicit RL framework	No explicit support- can be built and deployed on CloudML	Azure Personalizer	SageMaker RL
5.	Model Lifecycle Management	Machine Learning Accelerator	Not explicity, but AI Platform supports multiple versions of models being stored and served	MLOps	MLFlow
6.	Resource Monitoring	Cloud App Manegement- with cloud data collector, and infrastructure and monitoring reporting	Cloud Platform Moniroting, Stackdriver Monitor	Azure Monitor and Azure Resource Health	Amazon CloudWatch, SageMaker Model Monitor
7.	Training Visualization	Training can be monitored on the cmd line interface, from which graphs can be plotted.	Google Tensorboard to save and visualize data summary	Azure Tensorboard for visualisation and logs	No explicit support, but API calls from the model to the data can be monitored using CloudWatch
8.	Elastic Scaling	Allowed- included within the Watson ML Accelerator	Autoscaling can be specified for groups of instances based on resource usage of the VM.	Allows Azure scaling on Azure Compute instances and Kubernetes	Amazon Elastic Inference

Training job description- ie submitting jobs:

a) IBM Cloud Platform: Using training definitions. Additional parameters can be specified.

```
name: training-definition-1
description:  Simple MNIST model implemented in TF
framework:
  name: tensorflow
  version: '1.13'
  runtimes:
    name: python
    version: '3.6'
training_data_reference:
- name: MNIST image data files
  connection:
    endpoint_url: <auth-url>
    access_key_id: <username>
    secret_access_key: <password>
  source:
    bucket: mnist-training-models
```

```
    type: s3

execution:

    compute_configuration:
      name: v100x2
```

Hyperparamter tuning is also possible:

```
hyper_params = json.loads(open("config.json").read())

learning_rate = float(hyper_params["initial_learning_rate"])
training_iters = int(hyper_params["total_iterations"]

hyper_parameters_optimization:

    method:
      name: random
      parameters:
      - name: objective
        string_value: accuracy
      - name: maximize_or_minimize
        string_value: maximize
      - name: num_optimizer_steps
        int_value: 4
```

b) GCP:

```
training_inputs = {'scaleTier': 'CUSTOM',
    'masterType': 'complex_model_m',
    'workerType': 'complex_model_m',
    'parameterServerType': 'large_model',
    'workerCount': 9,
    'parameterServerCount': 3,
    'packageUris': ['gs://my/trainer/path/package-0.0.0.tar.gz'],
    'pythonModule': 'trainer.task',
    'args': ['--arg1', 'value1', '--arg2', 'value2'],
    'region': 'us-central1',
    'jobDir': 'gs://my/training/job/directory',
    'runtimeVersion': '1.14',
    'pythonVersion': '3.5'}
```

Hyperparamter tuning is also possible:

```
hyperparams = {
    'goal': 'MAXIMIZE',
    'hyperparameterMetricTag': 'metric1',
    'maxTrials': 30,
    'maxParallelTrials': 1,
    'enableTrialEarlyStopping': True,
    'params': []}
hyperparams['params'].append({
    'parameterName':'hidden1',
    'type':'INTEGER',
    'minValue': 40,
    'maxValue': 400,
    'scaleType': 'UNIT_LINEAR_SCALE'})
hyperparams['params'].append({
    'parameterName':'rnnCellType',
```

```
    'type': 'CATEGORICAL',
    'categoricalValues': [
         'BasicLSTMCell',
         'BasicRNNCell',
         'GRUCell',
         'LSTMCell',
        'LayerNormBasicLSTMCell'
        ]
})

# Add hyperparameter specification to the training inputs dictionary.
training_inputs['hyperparameters'] = hyperparams
# Build the job spec.
job_spec = {'jobId': my_job_name, 'trainingInput': training_inputs}

# Add hyperparameter specification to the training inputs dictionary.
training_inputs['hyperparameters'] = hyperparams
# Build the job spec.
job_spec = {'jobId': my_job_name, 'trainingInput': training_inputs}

job_spec = {'jobId': my_job_name, 'trainingInput': training_inputs}
```

c) Azure:

Define the parameters in a separate file, and subsequently submit job

```
sampling:
    type: random # Supported options: Random, Grid, Bayesian
    parameter_space: # specify a name|expression|values tuple for each parameter.
    - name: --penalty # The name of a script parameter to generate values for.
      expression: choice # supported options: choice, randint, uniform, quniform, loguniform, qloguniform, normal, qnormal, lognormal, qlognormal
      values: [0.5, 1, 1.5] # The list of values, the number of values is dependent on the expression specified.
policy:
    type: BanditPolicy # Supported options: BanditPolicy, MedianStoppingPolicy, TruncationSelectionPolicy, NoTerminationPolicy
    evaluation_interval: 1 # Policy properties are policy specific. See the above link for policy specific parameter details.
    slack_factor: 0.2
primary_metric_name: Accuracy # The metric used when evaluating the policy
primary_metric_goal: Maximize # Maximize|Minimize
max_total_runs: 8 # The maximum number of runs to generate
max_concurrent_runs: 2 # The number of runs that can run concurrently.
max_duration_minutes: 100 # The maximum length of time to run the experiment before cancelling.
```

```
src = ScriptRunConfig(source_directory = script_folder, script = 'train.py', run_config = run_amlcompute)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Creation of clusters is also possible:

```
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           max_nodes=4)
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True)
```

d) AWS SageMaker

```
algo = AlgorithmEstimator(
            algorithm_arn='arn:aws:sagemaker:us-east-2:012345678901:algorithm/my-algorithm',
            role='SageMakerRole',
            train_instance_count=1,
            train_instance_type='ml.c4.xlarge',
            sagemaker_session=sagemaker_session,
            base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

All the platforms support batch transformations of data.

Set of common fields:
Instance Type (master and worker specs possible for distributed training), Hyperparameters, job name, job scripts, Framework and associated packages with versions.