ZDX GraphQL Best Practices

SEPTEMBER 2026

EBOOK



©2026 Zscaler, Inc. All rights reserved.

Table of Contents



Introduction 3

Rate-limit hygiene 4

Rate Limits 4

Transport 5

Query efficiency 6

Constraints 7

Sync patterns 8

Error Codes 9

Error handling 9

Python Examples 11

#1 Minimal: `requests` handles gzip for you 11

#2 Manual decompression: `stdlib` only (no requests) 12

#3 Production pattern: pagination + backoff + daily budget 13

©2026 Zscaler, Inc. All rights reserved.



Introduction

In modern digital enterprises, user experience is the primary measure of application and network performance. Zscaler Digital Experience (ZDX) provides end-to-end visibility into the complete digital experience of users across endpoints, underlay networks, Zscaler Zero Trust Exchange infrastructure, and SaaS/private applications.

As enterprise observability strategies mature, IT operations, SecOps, and SRE teams increasingly require deep programmatic access to this telemetry to fuel automated remediation, custom executive dashboards, ITSM workflows (e.g., ServiceNow), and enterprise data lakes (e.g., Splunk, Snowflake, Datadog).

To support high-velocity, high-cardinality telemetry extraction without the overhead of rigid endpoints, ZDX leverages a **GraphQL-based API architecture.** This guide provides architectural principles, operational best practices, and integration patterns to help engineering and operations teams maximize the performance, reliability, and value of their ZDX GraphQL integrations.

Base URLs

ZDX Standalow URL →

OneAPI →

©2026 Zscaler, Inc. All rights reserved.

Rate-limit hygiene

- Throttle client-side against the **daily** budget. Track consumption; alert at 80%.
- On 429: exponential backoff + jitter; honor `Retry-After` if present. Never tight-loop retry.
- One shared limiter per tenant credential. Multiple scripts on one API key starve each other invisibly.

Rate Limits

Tier	Licenses	calls/sec	calls/min	calls/hour	calls/day
1	5,000	5	30	1,000	10,000
2	20,000	5	60	3,000	15,000
3	40,000	5	120	6,000	30,000
4	100,000	12	300	15,000	75,000
5	> 100,000	12	450	22,500	150,000

©2026 Zscaler, Inc. All rights reserved.

Transport

- Send `Accept-Encoding: gzip` on every request. Fail fast in your client wrapper if the header is missing. Most HTTP libraries (Python `requests`, axios) add it and decompress automatically; raw sockets/curl scripts don't.
- Reuse connections (keep-alive/`requests.Session`); avoid paying TLS handshake per call.



©2026 Zscaler, Inc. All rights reserved.



Query efficiency

- Request only the fields you need. Full `Device` with all metric families when you want `profile` = wasted payload and server fan-out.
- Page at `first: 1000` (max) and walk `end_cursor` until `has_next_page: false`. **Note:** `end_cursor` returns **empty string** , not null, on the last page.
- Filter server-side (`loc_id`, `dept_id`, `geoloc`, `os_name`, `event_id`) instead of pulling everything and filtering in your code.
- Batch up to the 8-domain limit per query; one call carrying 8 apps' metrics, not 8 calls.
- `time_range` **sets your granularity, not just your scope.** The API is bulk in nature: metrics come back aggregated over the whole requested window. A 2-hour window returns one 2-hour aggregate i.e. you cannot recover hourly detail from it.
- **Want 24 hours of data? Pull 24 × 1-hour windows**, not one 24-hour call.
- Budget accordingly: hourly granularity for a 5,000-device fleet ≈ 5 pages × 24 windows = 120 calls/day.

©2026 Zscaler, Inc. All rights reserved.

Constraints

1. Time range: 1—48h span (`device_events` : 1—2h), 14-day max lookback, hour-aligned from/to, current hour not queryable.

2. Pagination: first 1—1000; `cursor` is Base64

3. Required selections:

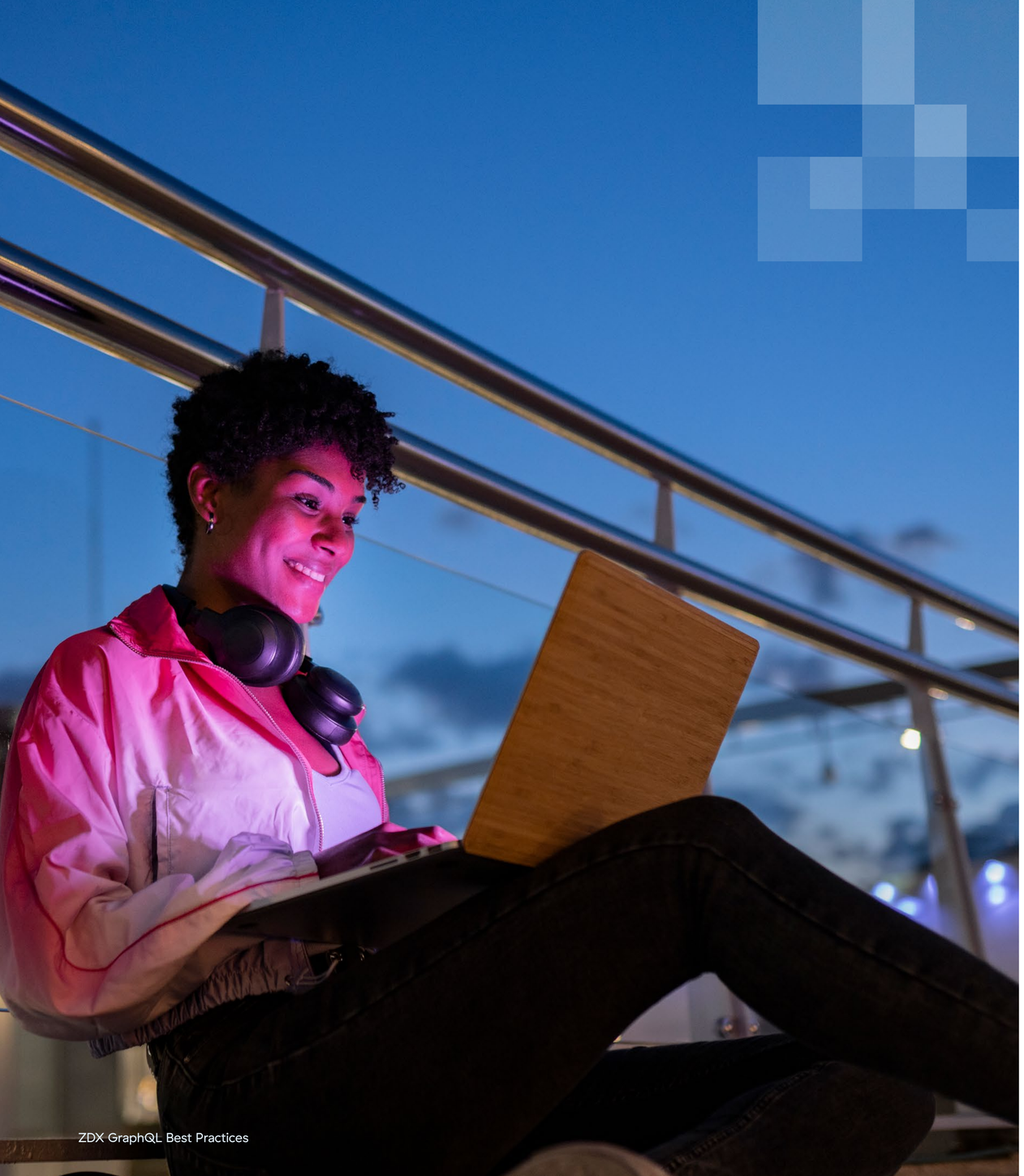
   - `total_count,`
   - `page_info {`
     `end_cursor`
     `has_next_page`
     `}`
   - `time_range`

4. Max 8 metric-node fields per devices query; event_id.in 1—5 entries; email filter ≤25 entries, valid format, in/not_in can't be identical

5. Query shape: max 4 aliases, complexity limit (say "complexity limit applies", not the 450/weight-2 internals), no duplicate fields with identical args, introspection disabled.



©2026 Zscaler, Inc. All rights reserved.



Sync patterns

- Incremental sync: poll the last completed 1-hour window each hour; dedupe on `device_id` + `timestamp`.
- Full-fleet resync nightly at most.
- Cache by volatility: `DeviceProfile` (hardware/OS) changes rarely; cache 24h. Health/app metrics don't cache.
- Run heavy jobs off-peak and spread them across the day; don't spend the whole budget at 9am.

©2026 Zscaler, Inc. All rights reserved.

Error Codes

Category	Code	Meaning	Client action
Auth	`UNAUTHENTICATED`	No/invalid token	Fix credentials; don't retry
Auth	`TOKEN_EXPIRED`	Token expired	Refresh token, then retry once
Auth	`FORBIDDEN`	No permission	Don't retry; check role
Auth	`INSUFFICIENT_SCOPE`	Missing OAuth scope	Don't retry; re-issue token with scope
Client	`BAD_USER_INPUT` / `INVALID_ARGUMENT` / `MISSING_REQUIRED_FIELD`	Bad query args	Fix the query; don't retry
Client	`VALIDATION_FAILED`	Business rule failed	Fix input; don't retry
Client	`CONFLICT` / `ALREADY_EXISTS`	Resource state conflict	Don't retry blindly
Client	`MISSING_MANDATORY_COMPRESSION`	`Accept-Encoding: gzip` header absent	Add header; don't retry without it

Error handling

- Check the GraphQL `errors` array **even on HTTP 200**, partial responses are valid GraphQL.
- **Always capture `x-request-id` from response headers** and include it in logs and support tickets; it's the key to fast debugging on the Zscaler side.
- Branch on error code, not message text. Retry only what's retryable (below).
- Use query variables; never string-interpolate values into query text.
- Log `total_count` vs edges returned. Persistent `total_count: 0` usually means a bad filter, not empty data.

©2026 Zscaler, Inc. All rights reserved.

Error Codes (cntd.)

Category	Code	Meaning	Client action
Data	`NOT_FOUND`	Resource doesn't exist	Don't retry
Data	`DATA_NOT_READY`	Data still processing	**Retry later**
Data	`RESOURCE_LOCKED`	Temporarily locked	**Retry later**
Data	`PRECONDITION_FAILED`	Required state not met	Don't retry until state changes
Throttle	`RATE_LIMITED`	Too many requests	**Retry** with backoff; honor `Retry-After`
Throttle	`RETRY_LATER`	Temporary condition	**Retry** with backoff
Server	`INTERNAL_SERVER_ERROR/ SERVICE_UNAVAILABLE/ TIMEOUT/ DEPENDENCY_FAILURE`	Server-side issue	**Retry** with backoff; escalate with `x-request-id` if persistent

Rule of thumb:

retry `DATA_NOT_READY`, `RESOURCE_LOCKED`, `RATE_LIMITED`, `RETRY_LATER`, and all server errors with exponential backoff. Everything else is a bug in your request; retrying wastes rate-limit budget.

©2026 Zscaler, Inc. All rights reserved.

Python Examples

#1

Minimal: `requests` handles gzip for you

`requests` sends `Accept-Encoding: gzip, deflate` by default and transparently decompresses. Set the header explicitly anyway so the mandatory requirement is visible in code.

Python

```python
import requests

API_URL = "https://api.zdxcloud.net/graphql"

session = requests.Session()  # keep-alive: reuse TLS connection
session.headers.update({
    "Authorization": "Bearer <token>",
    "Content-Type": "application/json",
    "Accept-Encoding": "gzip",    # mandatory
})

query = """
query ListDevices($from: DateTime!, $to: DateTime!, $first: Int!, $after: String) {
  ZDX {
    devices(first: $first, after: $after, time_range: {from: $from, to: $to}) {
      total_count
      page_info { has_next_page end_cursor }
      edges { node { id { value } name { value } profile { os_name zdx_version } } }
    }
  }
}
"""

resp = session.post(API_URL, json={
    "query": query,
    # 1-hour window: API aggregates metrics over the whole range requested
    "variables": {"from": "2026-09-09T00:00:00Z", "to": "2026-09-09T01:00:00Z",
                  "first": 1000, "after": None},
})

resp.raise_for_status()
data = resp.json()                  # already decompressed by requests
if data.get("errors"):              # partial errors arrive on HTTP 200
    raise RuntimeError(data["errors"])
# verify compression actually happened (optional sanity check)
assert resp.headers.get("Content-Encoding") == "gzip"
```


©2026 Zscaler, Inc. All rights reserved.

#2

Manual decompression: `stdlib` only (no requests)

For environments where you control the raw response (or to show what happens under the hood)

Python

```python
import gzip, json, urllib.request
req = urllib.request.Request(
    API_URL,
    data=json.dumps({"query": query, "variables": variables}).encode(),
    headers={
        "Authorization": "Bearer <token>",
        "Content-Type": "application/json",
        "Accept-Encoding": "gzip",   # mandatory
    },
)

with urllib.request.urlopen(req) as resp:
    raw = resp.read()
    if resp.headers.get("Content-Encoding") == "gzip":
        raw = gzip.decompress(raw)   # urllib does NOT auto-decompress
    data = json.loads(raw)
```


©2026 Zscaler, Inc. All rights reserved.

#3

Production pattern: pagination + backoff + daily budget

Python

```python
import time, random, requests

class ZdxClient:
    def __init__(self, token, url, daily_budget=10_000):  # tier 1 default
        self.s = requests.Session()
        self.s.headers.update({
            "Authorization": f"Bearer {token}",
            "Content-Type": "application/json",
            "Accept-Encoding": "gzip",
        })
        self.url = url
        self.calls_today = 0
        self.daily_budget = daily_budget

    RETRYABLE = {"RATE_LIMITED", "RETRY_LATER", "DATA_NOT_READY", "RESOURCE_LOCKED",
                 "INTERNAL_SERVER_ERROR", "SERVICE_UNAVAILABLE", "TIMEOUT",
                 "DEPENDENCY_FAILURE"}

    def post(self, query, variables, max_retries=5):
        if self.calls_today >= self.daily_budget:
            raise RuntimeError("daily API budget exhausted")
        for attempt in range(max_retries):
            resp = self.s.post(self.url, json={"query": query, "variables": variables})
            self.calls_today += 1
            req_id = resp.headers.get("x-request-id")  # include in every log/ticket
            if resp.status_code == 429:
                wait = float(resp.headers.get("Retry-After",
                             2 ** attempt + random.random()))
                time.sleep(wait)
                continue

            resp.raise_for_status()
            data = resp.json()
            errors = data.get("errors")
```

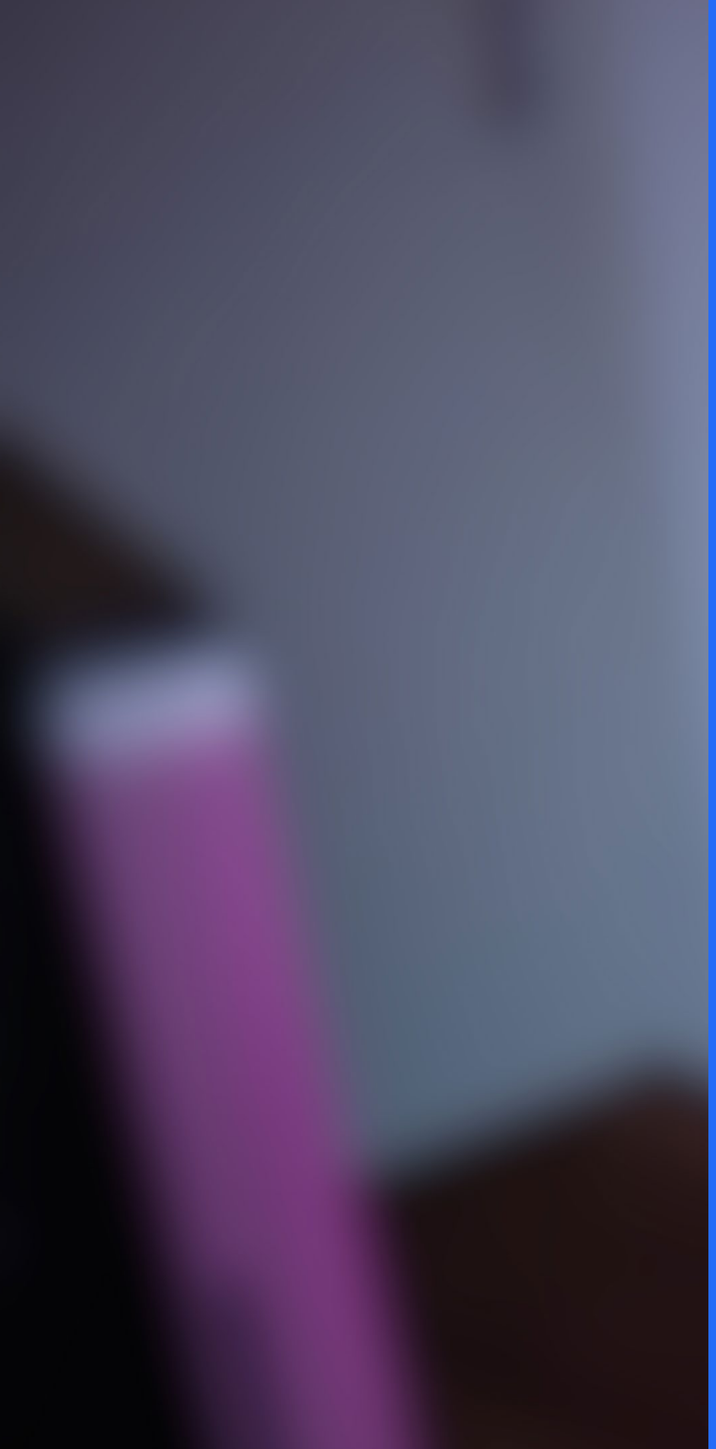

©2026 Zscaler, Inc. All rights reserved.

NOTES FOR DOCS READERS:

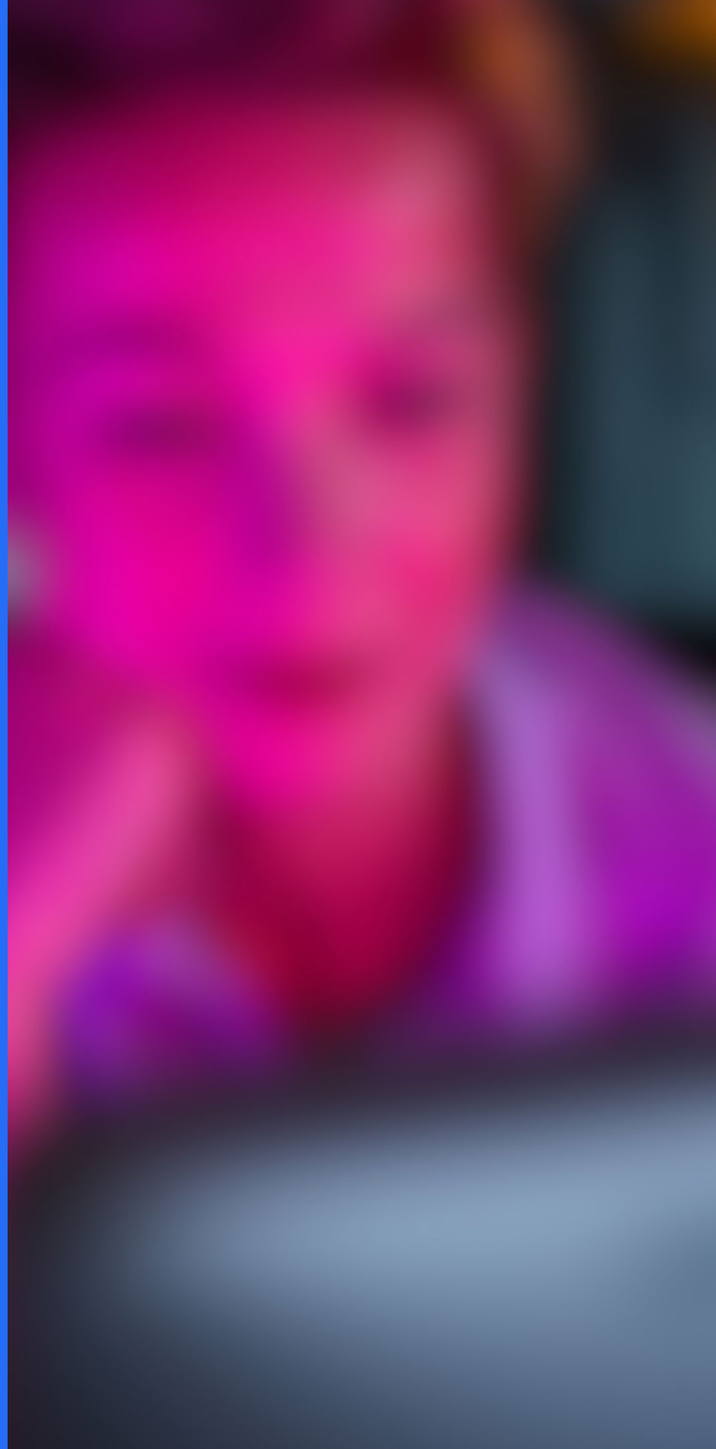
`requests` auto-decompresses gzip; if you log raw payload sizes, use `resp.headers.get("Content-Length")` (wire size) vs `len(resp.content)` (decompressed) to view the bandwidth win.

```
continued...

            if errors:
                codes = {e.get("extensions", {}).get("code") for e in errors}
                if codes & self.RETRYABLE:
                    time.sleep(2 ** attempt + random.random())
                    continue
                # non-retryable: bad query/auth — retrying wastes budget
                raise RuntimeError(f"GraphQL errors {codes} (x-request-id={req_id}): {errors}")
            return data["data"]
        raise RuntimeError(f"retries exhausted (last x-request-id={req_id})")

    def all_devices(self, query, variables):
        """Walk all pages. end_cursor is empty string (not null) on last page."""
        after = None
        while True:
            page = self.post(query, {**variables, "after": after})["ZDX"]["devices"]
            yield from (e["node"] for e in page["edges"])
            if not page["page_info"]["has_next_page"]:
                break
            after = page["page_info"]["end_cursor"]

client = ZdxClient("<token>", API_URL)

# 24 hours of data at hourly granularity = 24 x 1-hour windows.
# The API aggregates over the requested range; one 24h call = one 24h aggregate.
from datetime import datetime, timedelta, timezone

start = datetime(2026, 9, 9, tzinfo=timezone.utc)
for h in range(24):
    frm, to = start + timedelta(hours=h), start + timedelta(hours=h + 1)
    for device in client.all_devices(query, {"from": frm.isoformat(),
                                             "to": to.isoformat(),
                                             "first": 1000}):
        print(frm, device["id"]["value"])
```


©2026 Zscaler, Inc. All rights reserved.



zscaler™ | Act Fast. Stay Secure.



About Zscaler

Zscaler (NASDAQ: ZS) is a pioneer and global leader in zero trust security. The world's largest businesses, critical infrastructure organizations, and government agencies rely on Zscaler to secure users, branches, applications, data & devices, and to accelerate digital transformation initiatives. Distributed across more than 160 data centers globally, the Zscaler Zero Trust Exchange™ platform combined with advanced AI combats billions of cyber threats and policy violations every day and unlocks productivity gains for modern enterprises by reducing costs and complexity.

© 2026 Zscaler, Inc. All rights reserved. Zscaler™ and other trademarks listed at **zscaler.com/legal/trademarks** are either (i) registered trademarks or service marks or (ii) trademarks or service marks of Zscaler, Inc. in the United States and/or other countries. Any other trademarks are the properties of their respective owners.

+1 408.533.0288 **Zscaler, Inc. (HQ) • 120 Holger Way • San Jose, CA 95134** **zscaler.com**